ISSN 0265-2919

80p 53

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

SPECIAL NEEDS For the mentally and physically handicapped, the development of the microcomputer has opened new worlds — 1041

HARDWARE

ON THE BLOWER We explain the internal workings of an EPROM programmer, and take a look inside one from HCR Electronics — 1044

ICONOGRAPHIC ART The AMX mouse package provides the hardware and software to allow BBC Micro owners a Macintosh-like graphics facility — 1050

SOFTWARE

SCHOOL TURTLES As the combination of the BBC Micro and the LOGO language gain a steady hold in the education market, we look at two packages — from Acornsoft and Logotron — written for that machine — 1054

COMPUTER SCIENCE

CONSTRUCTIVE DECISIONS A discussion of several useful decision-making constructs available in PASCAL — 1046

PROGRAMMING PROJECTS

ALL ABOARD The first module of our 16th century trading voyage game involves setting up arrays of data about the crew — 1052

JARGON

FROM MAINFRAME TO MASS STORAGE A weekly glossary of computing terms — 1049

MACHINE CODE

BASIC INTERACTION Concluding our discussion of BBC BASIC and how it interacts with the machine's operating system — 1056

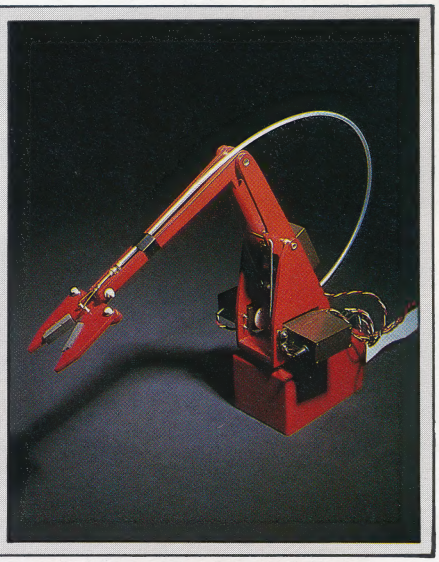
WORKSHOP

VISUAL AWARENESS We give our Workshop robot a primitive form of sight, using light-dependent resistors, and show how it can be made to follow a line — 1058

JOYSTICK JURY Our guest reviewer this week has been leading a very active life in front of the Spectrum — INSIDE BACK COVER

Next Week

- **We begin the most exciting and ambitious project yet in our Workshop series — the construction of a robot arm, which you will be able to interface with the BBC Micro, Commodore 64 and the Spectrum.**
- **Our PASCAL course continues with a look at arithmetical operations.**
- **In our vertical software series, we go down on the farm to see how computers are revolutionising agricultural management.**



QUIZ

1) The BBC Micro contains an eight-bit processor. Why, therefore, does a 16 Kbyte ROM need fourteen address lines?
2) The AMX mouse software produces a display very similar to the applications software on another micro. Which machine is it?
3) What function does the CASE statement perform in PASCAL?
4) Two of the light-dependent resistors (LDRs) on our Workshop robot measure the ambient light. What would happen when the robot moves into shadow, and why?

QUIZ

Managing Editor Mike Wesley; **Editor** Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Technical Editor** Steve Colwill; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Jonathan Kaye; **Contributors** Geoff Bains, Nick Walsh, Joe Pritchard, Steve Malone, Harvey Mellar, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Manager** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984; Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: Aus$1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY GEORGE LOGAN

SPECIAL NEEDS

We have seen how the computer enables pupils to carry out traditional tasks in a more efficient and rewarding way. For the handicapped, however, the micro is more than an enhancement to learning — it's a vital means of breaking down the barriers to the normal world.

Many adults and children have special needs in education. This may be because of a physical handicap such as deafness or blindness, or because of a mental handicap. While some needs are obvious, others are difficult to diagnose and understand and each disability presents unique problems, often brought about by the insensitivity of society rather than by the disability itself.

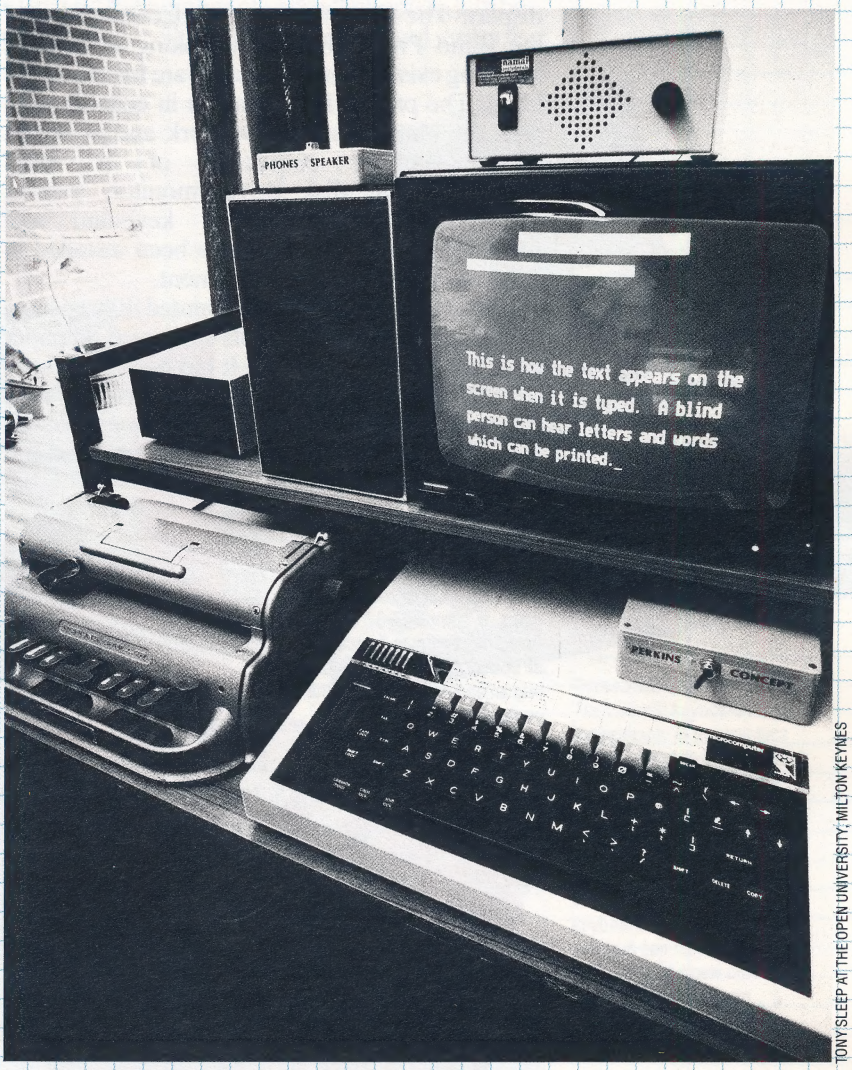
Sadly, government cuts hit what few special schools there are as hard as they hit mainstream education. Consequently this is not a lucrative market for hardware and software developers. Nevertheless, the energy and dedication of those working in this area can serve as an example to the rest of the education system. A group of volunteers in South Yorkshire, for example, raised £10,000 in their spare time to develop a special low-cost telephone for the deaf. Most of such work is done at 'grass roots' level by individuals and voluntary groups, and across the country school teachers, programmers and engineers are working on their own initiative, developing hardware and software for special education.

In Britain, Special Education Microelectronics Resources Centres (SEMERCs) have been set up in Manchester, Newcastle and Redbridge to meet the needs of special education. One of their main functions is to educate teachers in special schools and inform them of relevant developments. The resources of the four SEMERCs are small and their task is vast. The centre at Redbridge serves 700 schools and must divide its time between educating staff, giving advice and information, and evaluating and developing hardware and software.

The situation is made more complex by the many different problems encountered in special education. With some disabilities, for example, it is communication that presents the greatest problems. An intelligent, alert person can be trapped inside a disabled body, unable to communicate with the outside world. Microtechnology has opened new channels of communication for many people. For example, the Photonic Wand, an optical sensor mounted on a plastic helmet and controlled by head movements, enables people who are unable to speak and have no control of their limbs to operate a BBC Micro. The wand connects to the computer via the analogue input port. The optical sensor is similar to a light pen and moves a cursor across the screen in response to head movements.

A word processing program called Write can be used with the wand. It displays the alphabet on the screen and individual letters, in upper or lower case, are selected by pointing. There is an edit function and word lists are available. Text can be printed, saved or deleted. A program called Paint allows drawing with six colours and another, called Music, displays a keyboard. Notes are selected with the cursor and played.

The wand can be used by people with unsteady hand movement and children as young as eight. Work has been done at Manchester SEMERC to connect the Photonic Wand to a speech synthesiser. Responses designed for telephone



TONY SLEEP AT THE OPEN UNIVERSITY, MILTON KEYNES

Place Of Work
The Vincent work station combines a number of modifications, making it useful to people with different disabilities. The Perkins Brailler enables blind people to input information to the computer, and the speech synthesiser can give audible feedback. The text display on the VDU has been specially enlarged for the benefit of the partially sighted and the work station also connects to a concept keyboard. Although the work station illustrated here has an ordinary printer attached, a special Braille printer is also available, but at £3,000 may be beyond the means of many users

conversations, such as 'can you repeat that phrase', are selected on the screen and spoken by a D E Systems speech synthesiser. It is hoped to expand the vocabulary and to develop a simple way of connecting the synthesiser to the telephone.

Computers are designed for the sighted. Monitors, printers and plotters are all visual outputs, and all programs make use of visual display. The Open University's Computing and the Blind Project has been looking at ways of adapting existing low-cost hardware for use by the blind. The project has been run in collaboration with the blind at school, at work and at home. Work stations consisting of a BBC microcomputer, disk drive, monitor, printer, speech synthesiser, concept keyboard and modified Perkins Brailler have been installed in schools and places of employment.

The Perkins Brailler was invented 40 years ago to type braille, a code of dots embossed on paper to enable reading by touch. It has been interfaced to the computer to enable a blind person to read the printout. Software is available to enable the computer to transcribe braille into normal text and to store, edit and print it.

Magic Touch
The photonic wand enables even severely handicapped people, unable to speak or move their limbs, to control a BBC Micro. Wand movement is detected by an optical sensor and translated into an analogue signal for the computer to process

Several talking word processors have been developed for use with the standard keyboard. Typed characters and special keys are verified by spoken output. The Delete key also speaks the name of the character deleted. Text is edited by using a speech cursor. As it moves through the text, characters, words or sentences are spoken. The cursor can be stopped for additions or deletions. The more advanced programs have facilities for formatting the text before it is printed. Margins, headings and spaces are defined to produce a neat final copy, indistinguishable from that produced by a sighted person.

Some handicapped children are not physically co-ordinated enough to use a standard keyboard. A flat, touch-sensitive pad has been developed with such people in mind. Called a 'concept keyboard', it allows interchangeable overlays designed by the teacher to be placed over it. The keyboard might, for example, be divided into four sections and a picture drawn on each quarter, enabling control of a floor turtle. Pressing different sections could send the turtle forwards, back, left or right. This is simpler than typing FD 1 RETURN. Some software and an authoring language, Starset, are available for the concept keyboard.

The 'mouse' used with the Apple Macintosh computer also avoids the 'keyboard barrier' and has great potential for special education. A user group, AMASE (Apple Macintosh Applications in Special Education), exists to exploit the possibilities of the machine.

Special switches, operated by various parts of the body, have been developed. For example, a switch has been designed for people with no

CHRIS BARKER, COURTESY OF EDUCATIONAL COMPUTING MAGAZINE

voluntary movement. Two small metal discs placed on the skin next to the eyes detect horizontal movement of the eyeball and the electrical signals are amplified to control switches.

At the Department of Science and Society at Bradford University a flashing orange light was attached to a turtle to enable a partially sighted, physically handicapped 18-year-old see it. Using a concept keyboard, he was able to move the turtle around the floor and to see the results of his actions. For this boy, as for many handicapped children, experiences like this often represent the first step taken from being a passive observer of their environment to consciously exerting an

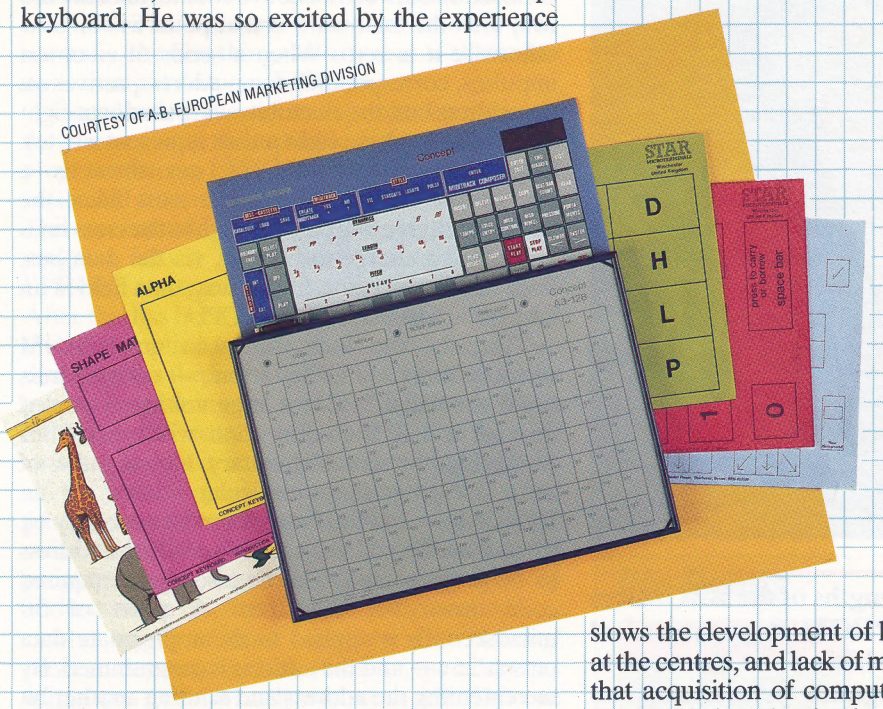
influence on their surroundings.

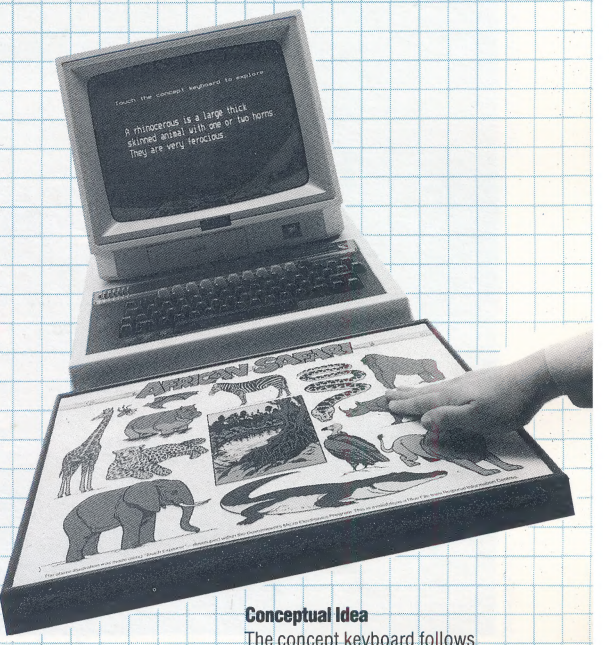
At the 1984 British Logo User's Group conference, Dr. Sylvia Weir gave remarkable examples of the progress that mentally and physically handicapped children made using LOGO and turtles. Michael, aged 17, in a special school, had never written a word. His teachers suspected he was intelligent but had never communicated it. After being introduced to the computer he was soon programming for 10 hours a day. Two years later he was at university writing a paper on 'Trapped Intelligence'. A 7½-year-old autistic boy was allowed to control a turtle from a button box, a device similar to the concept keyboard. He was so excited by the experience that he began to speak for the first time.

Dr. Weir has also been using LOGO at a special school with a group of mentally alert teenagers suffering from cerebral palsy. Controlling the computer has helped them overcome the passivity imposed upon them by their handicap. Dr. Weir wrote: 'The resulting dramatic improvement in sense of personal worth and the consequences for intellectual activity has led the school to implement its own computer centre using a recently implemented computer system'.

A joint effort by Manchester SEMERC and a local computer club produced Micromike, a device to help children with speech impediments. A modified CB radio microphone that connects to a BBC microcomputer, it allows children to control various activities on the screen using their voices. Software developed for it includes City, which enables them to draw a city skyline. The height and width of the buildings are determined by the volume and duration of the voice. Other programs allow stars to be shot out of the sky, a boat to be steered through rapids, and a helicopter to rescue a canoeist. They give children an opportunity to control their voice in a unique way.

Like mainstream education, special education is chasing behind technological development. It is deprived of funds for purchasing and little research and development is done on its behalf in the commercial world, for software and hardware companies make their money from business and industry rather than education, and the special education market is even smaller. The SEMERC centres cannot cope with existing requests for help, and the number of special schools in any one area is huge. Shortage of money and expertise slows the development of hardware and software at the centres, and lack of money in schools means that acquisition of computer technology is slow. Despite being deprived of resources, however, exciting work is in progress.





Conceptual Idea
The concept keyboard follows the principle of the graphics tablet. Lowering the resolution of the board and adding an overlay allows even physically uncoordinated users to interface effectively with the computer without having to master the QWERTY keyboard. The design of the overlays can help reduce the reluctance of a new user to approach the computer

Computers And The Gifted

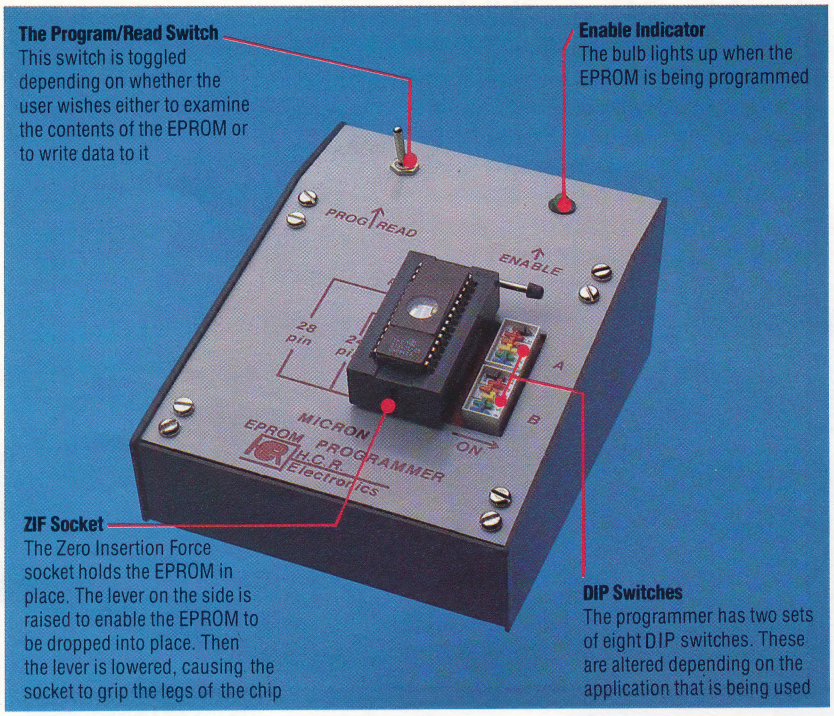
Gifted and talented children also have special educational needs, and the use of micros in education provides an exciting outlet for them as well as for handicapped children. In fact, many children with physical handicaps or perceptual disabilities, such as dyslexia, are gifted intellectually.

As is true for the handicapped, the microcomputer provides the gifted child with a means of gaining control over the environment and overcoming natural or man-made barriers to individual growth and self-actualisation. Specifically, computers can offer the gifted child:

- The opportunity to develop skills and interests at his or her own pace;
- Creative new ways to approach problem solving;
- A forum for communication and co-operation with other gifted students;
- The opportunity to practice tedious and tiresome exercises in a way that fends off boredom and maintains their interest;
- A new medium for, and field of, discovery

ON THE BLOWER

COURTESY OF TEXAS INSTRUMENTS



CHRIS STEVENS

Blower Parts
An EPROM programmer allows users to keep their programs permanently in memory. Programs are loaded into the computer and then transferred to the programmer via the user port where they are 'blown' onto the chip

One of the great strengths of the BBC Micro is its facility for EPROM chips (priced at around £5 each), which allow the user to keep important programs permanently on board the computer. The Micron EPROM programmer — or 'blower' as it is popularly termed — is an inexpensive device for programming EPROM chips.

One of the biggest advantages of the BBC Micro is its flexibility. In designing the machine, Acorn chose to include four empty ROM slots, allowing the user to place extra ROMs for specific applications in the machine, and these can be 'paged' by use of the * command. The provision of these slots has been one of the greatest successes of the machine, since it enables users to have programs fitted on-board that can be accessed much faster than having to LOAD them from tape or disk.

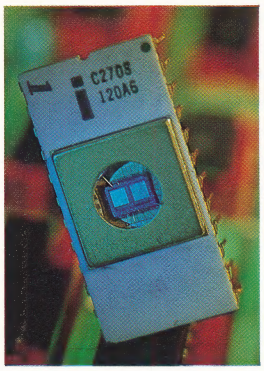
There is now an enormous selection of these applications ROMs available, covering a wide range of programs, including word processing, databases, and languages such as LOGO. However, it is also possible for users to write their own programs and then transfer them to an EPROM (erasable programmable read only memory), allowing them to be permanently stored within the machine. This is done by programming the EPROM with a device known as, naturally enough, an 'EPROM programmer', which is popularly known as an 'EPROM blower'.

Like all ROMs, an EPROM consists of a matrix of electrical lines in columns and rows, each intersection of which is known as a cell. If there is a connection between a line in a column and one in a row, then the logic state at that point will be one, otherwise it is zero. When a series of electrical impulses, corresponding to an address, arrives on the address bus, the electricity will pass down eight of these lines (these eight pulses represent a byte). Depending on whether or not there is a connection on the appropriate cross lines, the electrical charge will pass through to the cross lines and cause a charge to be sent, via the data bus, back to the processor. In this way, data held at an address is transferred to the CPU.

Typically, the logic states in an EPROM memory array are all set at one — each address holds the value &FF. If a large voltage is passed through a cell, the connection will break, thus producing the complementary logical state of zero. To program a particular address of an EPROM, we have to first set up the necessary bit pattern on the address pins, to specify the address to be programmed. Then we send the required code down the data bus. A pulse of 50 microseconds is applied to the appropriate data pins at 25v — and the address is programmed. By incrementing the address and sending another bit pattern to the data pins, it is possible to program the entire EPROM. However, unlike some other types of programmable ROMs in which this process is irreversible, an EPROM, as its name suggests, can be wiped clean.

An EPROM is characterised by a small quartz window above the chip. When strong ultraviolet light is shone through this window, the atoms in the chip are ionised to a higher electrical state and the bonds between the atoms will be re-established. This allows current to pass once more between the cross lines on the matrix, and the logical state will again be one. Ultraviolet light is not strictly necessary, however, since the EPROM programmer can write a software routine to reset all the memory locations to &FF.

The Micron EPROM programmer from HCR Electronics enables the user to program up to 16 Kbytes of machine code onto an EPROM. The programmer is a small box, the most prominent feature of which is a ZIF (zero insertion force) socket. This enables EPROMs to be inserted without damaging the pins which, once broken, render the chip useless. The ZIF socket is designed to accommodate either 24-pin or 28-pin chips. Two sets of DIP switches, which are altered



Close Examination
The chip within the EPROM housing can be easily viewed through the quartz window in the centre. Careful examination of the chip will show the thin pieces of wire that join the chip I/O connections to the pins. A closer look will show the arrays that make up the EPROM

depending on which of the various options one uses, are beside the ZIF socket. On the top left-hand side of the programmer is a switch that needs to be set according to whether you want to program or read the EPROM. Obviously, care has to be taken when using the programmer to ensure that the switch is on the correct setting — otherwise the EPROM may be reblown, thus losing the program held on the chip.

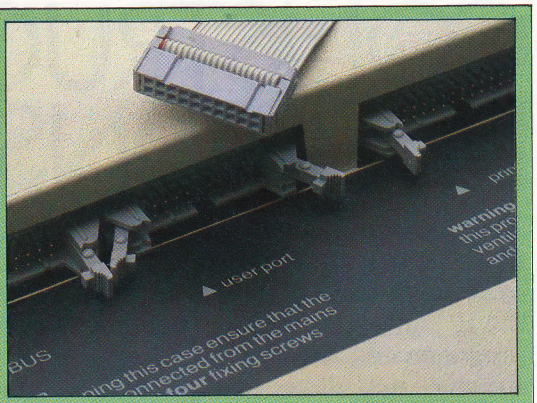
The device has a ribbon cable that is connected to the BBC's user port from the power supply. You can LOAD and SAVE programs into a buffer area, and then transfer the programs onto the EPROM with the menu-driven software provided with the machine. The software is held on cassette and, naturally, the first thing the new user is advised to do is dump it onto an EPROM.

Once loaded, the software presents a menu on the screen giving the available options. To copy a program onto an EPROM, the program must first be loaded into memory using the (L)oad option. This command has the effect of placing the program into a buffer area, set aside by the EPROM software, starting at location &2000. After this has been done, the user then selects the (P)rogram option from the main menu. The screen then displays the list of EPROM types that can be programmed using the device. After choosing an EPROM the screen will display the 16 DIP switches and their settings; switches needing to be altered from the previous program will flash. The program prompts the user to flick the read/write switch to PROG. Pressing any key will initiate the programming of the EPROM, with each memory address being displayed as it is programmed.

The length of time taken by the 'blowing' process depends on how many of the addresses within the EPROM will stay set at &FF. This can take anywhere between 74 seconds for a short program, to a maximum of 14 minutes for a 16 Kbyte EPROM. By using the (V)erify command, you can check whether the EPROM has been programmed correctly. The software computes a checksum, and any faulty addresses are listed on the screen. If there are no errors, the EPROM can then be safely fitted into one of the ROM sockets of the BBC Micro.

Standard headers can be fixed to a program. Using the G command, a routine can be added to the beginning of the program, attaching a name, which can later be called with the * prefix, and a CALL command. When loading BASIC programs onto the EPROM, this command is used in conjunction with the (F)ill option. This command ensures that addresses not used by the BASIC program are left set at &FF and are hence still programmable. Headers can also be affixed to machine code programs but these can only be loaded from address &2000.

Considering how useful these devices are, it is perhaps surprising that EPROM programmers have not become more popular with BBC Micro owners. It is likely that many users feel these programmers belong to the realm of 'serious' electronics enthusiasts and beyond the scope of most home computer users. This is simply not so. EPROM programmers are easy to use, no knowledge of electonics is assumed, nor necessary, and the machines offer a number of advantages. Primary among these is speed of access — a program can be called instantly via the operating system without the need to LOAD from disk or cassette. Furthermore, once the programs are incorporated into the computer, the applications held on the EPROM can be called from within BASIC programs. Finally, users are given the immense satisfaction of seeing their custom-built programs taking their places among the BBC Micro's own on-board programs.



Interface Connections
The EPROM programmer connects, through a 12-way ribbon cable, to the BBC Micro's user port

EPROM BLOWER

PRICE
£53.47 (including VAT)

DIMENSIONS
140 x 105 x 65 mm

INTERFACES
Ribbon cable connector that plugs into the BBC Micro's user port

DOCUMENTATION
Three photocopied sheets that explain each of the nine commands available for the machine. Although the machine itself is easy to use, the documentation is pitched a little high for the beginner

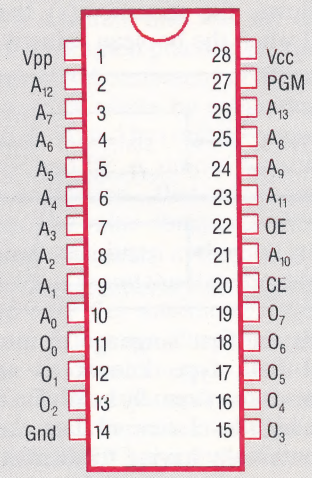
STRENGTHS
The machine is easy to use. The menu-driven software requires no great understanding from the user of the workings of the machine in order to operate it

WEAKNESSES
Little explanation as to how the user might develop the use of the machine beyond following the software instructions. Machine code programs that cannot be re-compiled to begin at &2000 cannot be easily transferred to EPROM

Pin Points

Pin Names	
A_0-A_{13}	Addresses
CE	Chip Enable
OE	Output Enable
O_0-O_7	Outputs
PGM	Program
Vpp Vcc	Supply Voltages
Gnd	Earth

This is a diagrammatic representation of the TMS27128 16 Kbyte EPROM, one of the chips that can be 'blown' using the HCR programmer. This is a 28-pin device (although it will also accept 24-pin chips). The chip has 14 address pins that allow it to address the 16 Kbytes of memory. The TMS27128 also has eight pins to provide an eight-bit output



LIZ DIXON

CONSTRUCTIVE DECISIONS

Our PASCAL course now turns to the subject of decision-making constructs. We consider the IF statement, which is familiar to BASIC programmers, and the CASE statement, a structure that allows multiple choices to be made within one construct.

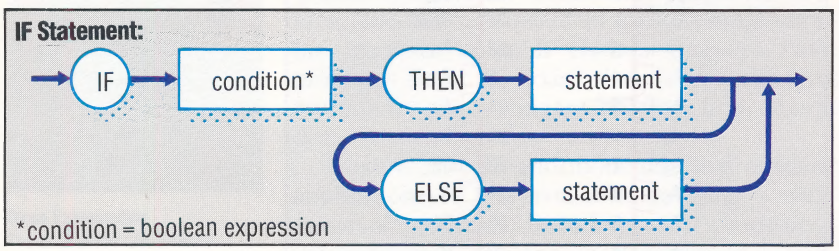
The IF statement in PASCAL is similar to that in most languages. Because the end of a line does not terminate a statement, we can use PASCAL's free format to show the possible paths through the structure of the IF statement with logical indentation. Here are two IF statements:

```
IF count = limit
   THEN
      WriteLn ('No room')
   ELSE
      write ('Next ?');
IF number > maximum THEN
   maximum := number
```

In the second example, the absence of an ELSE clause implies:

```
   ELSE
      {do nothing}
```

The alignment of the reserved words THEN and ELSE (when present) helps your eye to follow the flow of control through the 'construct'. A semi-colon is used to separate statements, as shown in the example, and so they can *never* appear before an ELSE. Remember that there is no ELSE statement in PASCAL only an IF statement; THEN and ELSE are used to delimit the boolean condition and the statements in the two clauses. In fact, every time we use an IF statement, we are testing a boolean value. The THEN 'clause' is executed only if the expression evaluates to true and, if an ELSE clause follows, the statement(s) there will be performed when the boolean value is false.



PASCAL was the first language to introduce the conceptual data type known as enumerated scalars. These are extremely helpful in enabling us to retain a high level view of data classification, without continually having to mentally translate data into numeric codes. (Numbers are the computer's language, not ours — unless we happen to be solving a specifically mathematical problem.) We have already seen how a simple constant definition can help us here:

```
CONST
   width = 80;
```

The constant identifier width may then be used throughout the program to reference the number of columns across the width of the screen or printer. Rewriting the program for a 40-column VDU is then a simple matter of changing one line in the definition at the start of the program; all the calculations for print formatting and so on will be automatically altered accordingly. A complete scale of constants could then be defined in this way.

If we were using colour graphics, the colours available could be mapped onto an appropriate scale of numbers (red = 1, green = 2, for example), but this would theoretically accommodate taking the square root of green, or multiplying blue and yellow! This is not only illogical, it is also a potential source of error when we are reasoning about problems that do not in themselves deal with numeric data. The TYPE definition part of a PASCAL program can be used to define an entirely new conceptual scalar type by simply enumerating a list of identifiers representing all the constant values of the scale. For example:

```
TYPE
   hue = (red, green, yellow, blue, magenta, cyan);
```

Because it is a definition (of the new type) and not a declaration, the syntax uses the equals sign to define the type identifier (hue) to refer to the ordered values of the type enclosed in brackets. Analogously, the pre-defined type integer refers to all the whole numbers available on the PASCAL implementation.

As always in PASCAL, successive identifiers in a list (in this case, colour values) are separated from each other with commas. These values are, of course, internally mapped onto integer values with the ordinal numbering starting at zero. This numeric representation is automatically organised by the compiler, and is much the same as the underlying codes for the computer's character set. Just as the character A has an ASCII code of 65, each value of hue will have an ordinal number that we can obtain with the scalar function ord. So, in this example, ord (red) is 0 and ord (cyan) would be 5. Having now defined this new scalar type, we could declare a variable in the usual way:

```
VAR
   colour : hue;
```

This declares an identifier (colour) to be a data item of the type hue, just as the declaration:

```
VAR
   letter : char;
```

indicates the character nature of the data object named letter. The only operations defined on enumerated types are relational tests and the use of the scalar functions. For example, we could write:

```
if colour < cyan then
   colour := succ (colour)
```

Remember that pred (red) and succ (cyan) don't exist! The variable colour is incompatible with variables of any other type, scalar or otherwise. This means that we can no longer perform illegal functions like taking square roots or saying:

```
colour := colour + 1
```

This does lead to one obvious restriction. Characters and numbers can be used as parameters for write and WriteLn statements, but

```
WriteLn (colour)
```

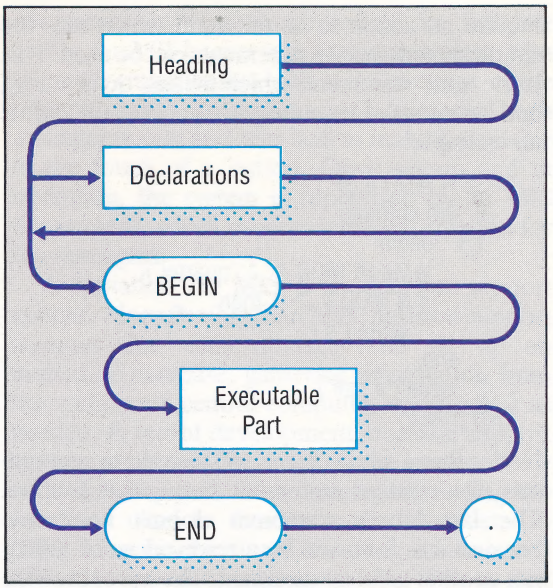
would be illegal. This is because the values of the type are purely conceptual, and if we want to print their names, we must map the colour values on character strings. This is an ideal application for the other PASCAL choice construct, the CASE statement.

We have seen how the IF statement in PASCAL looks entirely familiar, and benefits in readability from PASCAL's free format conventions. There are times, however, when multiple option decisions have to be taken that would need something like:

```
IF N = 1
   THEN
      write ('st.')
   ELSE
      IF N = 2
         THEN
            write ('nd.')
         ELSE
            IF N = 3
               THEN
                  write ('rd.')
               ELSE
                  write ('th.')
```

Whenever the statement to be executed depends on the value of a simple scalar being in a certain limited range, we can use the CASE statement to advantage. In this instance:

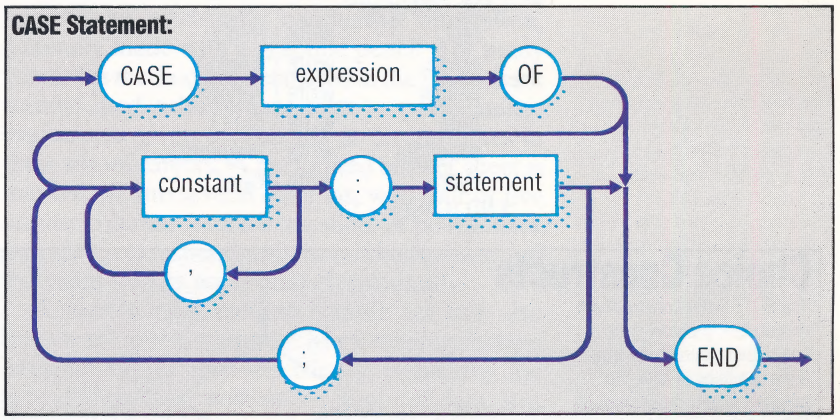
```
CASE N OF
   1          : write ('st.');
   2          : write ('nd.');
   3          : write ('rd.');
   4,5,6,
   7,8,9      : write ('th.')
END {CASE}
```



Skeleton Program
This is simplified syntax diagram of a PASCAL program, showing all the major elements. Note that PASCAL reserved words appear in upper case. This is a convention developed to set them apart from non-reserved words. Most versions of PASCAL, however, will accept reserved words in lower case

Notice that this is satisfactory only for values of N in the range of 'case labels' specified in the 'body' of the CASE statement — 1 to 9 in this example. All the values that N might have at execution time *must* be listed individually, and it would be illegal, for example, if N = 0 were entered.

Many PASCAL compilers possess the additional reserved word OTHERWISE or OTHERS, which may be used to list a default action. Refer to your manual for the extended syntax necessary here.



LIZ DIXON

This is the only statement in PASCAL that uses END to delimit a structure that was not entered via a BEGIN, so it is common practice, and good style, to qualify every CASE statement's END as shown.

The construct works by evaluating the scalar expression delimited by (or between) the reserved words CASE and OF. A simple variable name is a minimal expression that does not require calculation. The value obtained is then compared with each constant listed in the case label lists, and when a match is found the statement following the colon, and *only* this statement, is executed. The flow of control does not 'drop through', therefore, so that the structural integrity of the construct is preserved. If several operations need to be performed, a compound statement enclosed between a BEGIN/END pair may of course be used.

In certain circumstances, some values might

require no action to be taken, in which case the simplest of all PASCAL statements can be used. This is the 'null' statement, which means 'do nothing', and it consists of absolutely no syntax at all! Here is an example:

```
CASE N MOD 4 OF
  0  :{do nothing};
  1,3  : begin
           write (N MOD 4 : 1, 'quarter');
           if N MOD 4 > 1 then
             write ('s')
         end;
  2  :write ('a half')
END {CASE}
```

Notice that a semicolon is still needed to separate this non-existent statement from what follows. The last label's statement doesn't need one because it is followed by a reserved word (END); not another label or statement. The use of the MOD operator in the expression ensures that the value must be within the range 0 to 3. MOD gives the remainder from integer division, as in BBC and other BASICS.

In the next instalment of the series, we will examine these and all the other PASCAL operators, as well as in-built functions. In the meantime, here's the solution to the problem of how to print the character strings for each value of our own colour type and hue:

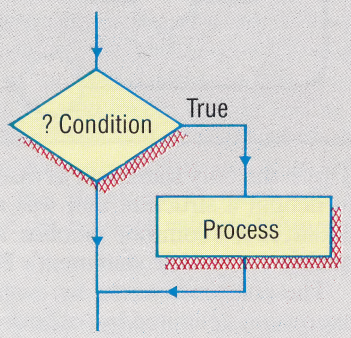
```
CASE colour OF
  red          : write ('Red');
  green        : write ('Green');
  yellow       : write ('Yellow');
  blue         : write ('Blue');
  magenta      : write ('Magenta');
  cyan         : write ('Cyan');
END {CASE}
```

Choice Constructs

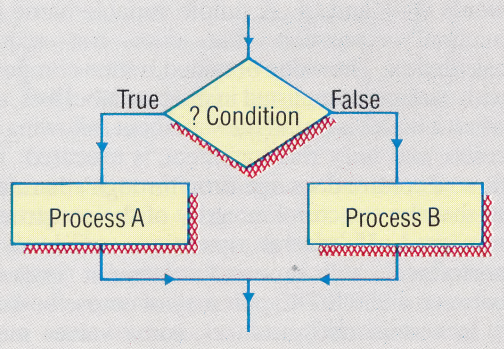
The IF . . . THEN construct will execute the 'process' if the condition is found to be true. If the condition is false, the program simply drops through to the next statement.

Whereas the IF . . . THEN construct presents an 'option', IF . . . THEN . . . ELSE presents a 'choice'. Depending on the outcome of the test performed at the beginning of the construct, either one of two processes will be executed

Option:
IF condition
THEN process



Choice:
IF condition
THEN process A
ELSE process B



Monthly Statements

This program, which simply calculates the number of days between a date input by the user and the end of the month, provides some useful examples of PASCAL programming principles.

The CASE statement is used to map numeric input data on to PASCAL enumerated types, using these for internal processing and then mapping back on to string output data. Notice the use of the dummy char variable (symbol) to read whatever delimiting character separates the input numbers

```
PROGRAM          Date      ( input, output);

CONST
    FullStop  = '.';

TYPE
    Calendar = ( Jan, Feb, Mar, Apr, May,
       Jun, Jul, Aug, Sep, Oct, Nov, Dec );

VAR
   Monthname     : Calendar;
   day,
   month,
   year,
   Left          : integer;
   symbol        : char;
   LeapYear      : boolean;

BEGIN
  WriteLn ( 'Enter the date in the form :' );
  WriteLn ( 'DD/MM/YY' : 40 );
  WriteLn;
  write ( 'Date ? ' );
  read ( day, symbol, month, symbol, year );
  LeapYear := year MOD 4 = 0;
                ( neglecting centuries )

  IF ( month > 0 ) AND ( month <= 12 )
    THEN
      CASE month OF

        1       : MonthName := Jan;
        2       : MonthName := Feb;
        3       : MonthName := Mar;
        4       : MonthName := Apr;
        5       : MonthName := May;
        6       : MonthName := Jun;
        7       : MonthName := Jul;
        8       : MonthName := Aug;
        9       : MonthName := Sep;
        10      : MonthName := Oct;
        11      : MonthName := Nov;
        12      : MonthName := Dec

      END       ( CASE )
    ELSE
      BEGIN
        WriteLn ( 'Eh ?');
        WriteLn ( '    - the program is ',
                  'about to CRASH !' )
      END;      ( MonthName is uninitialised )

    CASE MonthName OF

    Jan, Mar,
    May, Jul    : Left := 31 - day;

    Apr, Jun,
    Sep, Nov    : Left := 30 - day;

    Feb         : IF LeapYear
                    THEN
                      Left := 29 - day
                    ELSE
                      Left := 28 - day

  END; ( CASE )

  WriteLn;
  write ( 'There are ', Left : 1,
                'days left in ' );

  CASE MonthName OF

       Jan      : write ( 'January' );
       Feb      : write ( 'February' );
       Mar      : write ( 'March' );
       Apr      : write ( 'April' );
       May      : write ( 'May' );
       Jun      : write ( 'June' );
       Jul      : write ( 'July' );
       Aug      : write ( 'August' );
       Sep      : write ( 'September' );
       Oct      : write ( 'October' );
       Nov      : write ( 'November' );
       Dec      : write ( 'December' );

  END; ( CASE )

  WriteLn ( FullStop )

END .
```

M

MAINFRAME

Until the microprocessor revolution, all computer systems were extremely large and cumbersome. Then, the term *mainframe* simply meant the CPU and the memory of such a system. A few distinctions have since developed to indicate differences in size and theoretical capacity of computers, so the word mainframe has taken on a slightly different meaning.

The smallest levels are occupied by the microcomputers: pocket size, lap-held, portable, and desktop varieties. The next size up usually pertains to a multi-user system, in which several machines are connected into a cluster and can be used to manipulate the same data simultaneously. Above this level is the minicomputer, a high-powered multi-user system, usually consisting of a large central storage device, in the 20 to 40

megabyte range, and a 32-bit central processor. These have recently been augmented by faster systems with larger memories called 'super-minis'. Mainframes, the next step up, are still the room-sized systems used for data processing in large businesses, and cost around £1,000,000.

The term mainframe now applies to the entire system, not just the CPU and memory. When referring to mainframes, one usually means IBM or compatible machines, since IBM has captured about 70 per cent of the world market and is widely emulated. Yet these are not the biggest systems available, having lost that distinction to 'supercomputers' such as the Cray.

Spot The Computer
Mainframes are rarely the huge monolithic machines popularised by science fiction writers. The Cray-1, one of the newer breed of super mainframes, comes in several cabinet sized boxes — some of which could be mistaken for furniture!

MANAGEMENT INFORMATION SYSTEM

A *management information system* (MIS) is a data gathering and collecting system used by large companies. Its main purpose is to supply management with the necessary information when making business decisions. As an outgrowth of data processing, MIS developed as more companies became computerised in the late 1960s. An MIS begins with a systems analyst, who takes a comprehensive look at the structure of the organisation to determine what information each manager needs, and how best to make it available at the touch of a button. Once set up and in operation, the system is supervised by an MIS department, which processes the information for the managers.

A variation on MIS is a 'decision support system'. More flexible than MIS, decision support assumes that each manager will set up an individual database, gathering information from the company's central computer, and using it as needed. A recent development, decision support systems are largely the result of the increased use of microcomputers. More managers, feeling the increased need to directly access 'raw data' in order to make decisions, are using micro-based decision support to accomplish this. This in part explains why the MIS and data processing departments of large companies often feel threatened by the increased use of micros in business.

MARK SENSING

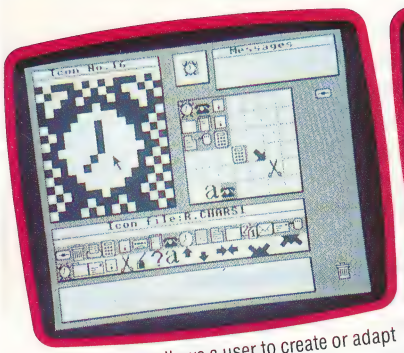
Mark sensing is the ability of a computer system to read a pencil mark or perforation and interpret the results as values or characters. Two methods can be used: one is technically labelled mark sensing, and the other, 'mark reading'. Mark sensing requires an electrically conductive impression, generally made with a graphite pencil. For many years, this method has been used in the US to mark standardised achievement tests — performance evaluations given to students at certain times in their school careers. A perforated answer sheet is marked off into several columns, with four or five answer columns for each question. A pencil mark is made to correspond with the correct answer, and the exam papers are scored electronically.

A more reliable method is mark reading, also known as 'optical scanning'. Mark reading systems interpret values photoelectrically, measuring the light and dark areas at each column of data. It is this method which is used to read the Universal Price Code, a series of black stripes that appears on product packaging and provides the product with a unique number. Optical scanners read the number, which describes in code what the product is and how much it costs.

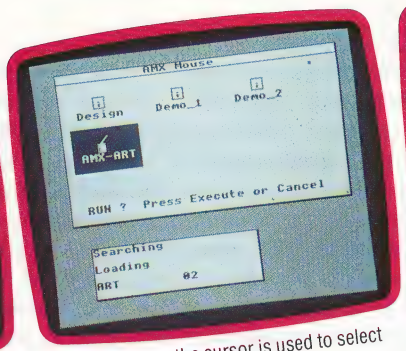
MASS STORAGE

Mass storage devices provide a system with a large amount of data storage space, larger than in conventional devices. In the early days of mainframe use, one megabyte — or one million bytes — was considered mass storage. But today, microcomputers like the IBM PC and Apricot xi have 10 megabytes of on-line storage, so this early distinction no longer applies. Mass storage devices are those like the IBM 3850, an automated tape library that can store up to 472 gigabytes (472,000,000,000 bytes) of data.

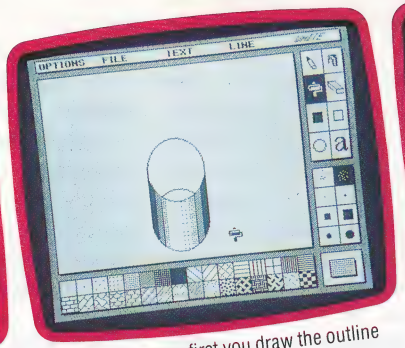
ICONOGRAPHIC ART



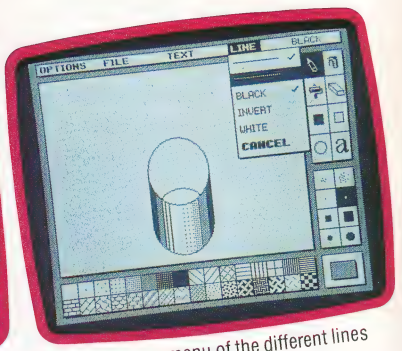
The icon editor allows a user to create or adapt the icon set



The Main Menu: the cursor is used to select one of the four applications programs



To create a picture: first you draw the outline



Selected from a menu of the different lines available

Mouse technology has been around a few years now, and most of the major manufacturers have mouse packages available for their systems. We look at the AMX mouse package, which includes the AMX Art program, designed specifically for the BBC Micro. With its simplicity of use, it is sure to become a popular item.

AMX Mouse

The AMX mouse comes as part of a complete package including the mouse itself, a ROM chip containing the driving software, applications software on cassette or disk, a user manual and an AMX Art applications manual. In execution, the package is very similar to that used on the Apple Macintosh, but is available to BBC Micro users at a fraction of the price

The AMX mouse package from Advanced Memory Systems, intended for the BBC Micro, comprises the mouse itself, along with a cable to connect it to the micro's user port, a ROM chip, two manuals, and cassette and disk software. Although both cassette and disk are provided, AMS freely admits that the AMX is clearly aimed at disk owners.

A standard mouse, made in Japan, the AMX is moulded in black plastic with a metal ball in its base for movement detection. The mouse fits comfortably into your hand, leaving the first three fingers resting above the three action selection buttons — Move, Execute and Cancel. The decision to incorporate a metal ball is a shame since it tends to slip a good deal on surfaces like polished wood and laminated desk tops. Therefore, placing a piece of paper on any slippery surface would be a good idea.

IAN McKINNELL

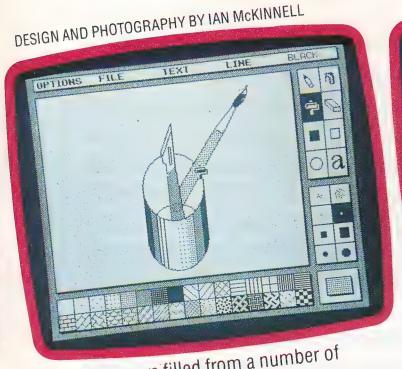
Inside the mouse, the ball is cradled by two rollers; at the end of each is a slotted disk that passes through a photoelectric cell. The disk breaks a beam of light, creating flashes that are picked up by the cell, and these are transmitted as directional commands to the computer. Thus, the computer knows exactly the location, movement and speed of the mouse at each moment.

The resident software in the sideways ROM accommodates communication with the computer. As well as providing the 'raw' routines for use by commercial software, this ROM also provides several easy-to-use routines enabling you to use the mouse on home-made programs. These include, amongst others, a routine to provide an updated position in screen co-ordinates of the mouse, one to read the selection buttons, and a routine to animate icons on the screen. By making use of these routines — all selected by using the * command structure of the BBC Micro's operating system — the user can quickly and easily write programs in BASIC that take full advantage of mouse control.

Routines are also provided to enable you to use the mouse on existing commercial software packages. The mouse movement can be configured to take the place of cursor keys, and the three selection buttons configured to act as any three keys on the keyboard, including the Shift and CTRL keys. In this way it is possible, for example, to use the mouse to delete, move and copy sections of text on the View word processor by Acornsoft.

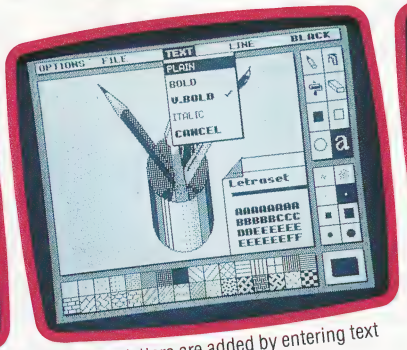
However, the beauty of the mouse is only really evident when using applications software

DESIGN AND PHOTOGRAPHY BY IAN McKINNELL

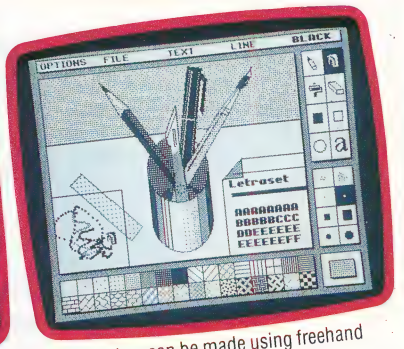


The outlines are filled from a number of different patterns . . .

And further textures are added



Words and letters are added by entering text mode and choosing the desired type



Final touches can be made using freehand mode

especially written for mouse control. The AMX package includes two such pieces of software that amply demonstrate the ease and naturalness of using the mouse. The first is a BASIC program for designing icons to incorporate into your own programs. Icons are graphics symbols that can be displayed in a program to indicate the menu choices or, alternatively, they can replace the cursor to indicate the selection made.

The Icon design package presents you with a grid of 16 by 16 squares in which you can design your icon by filling in the desired squares — all done with the mouse. Indeed, the only occasion you will need the keyboard with this program is to type a file name under which a group of icons is to be saved, or the file name of an existing set to be loaded. Because of the extent of the mouse's control, it is immensely easy to use, and designing complex and realistic icons is a quick and simple procedure. However, a better demonstration of the mouse's power is given by the AMX Art program, which is also provided. This is a full feature graphic design program entirely controlled by the mouse. It ranks with the Macpaint package on the Macintosh for ease of use, and in fact the AMX screen looks remarkably similar to Apple's creation.

USING THE ART PROGRAM

Drawing is performed in the central region of the screen — a Mode 4, 320 by 256, two colour display is used in the program. Down the right-hand side of the screen are two icon menus. The upper menu is used to select from a variety of drawing actions, including line drawing, erasing, spray paint effects and area filling, and they are represented on the screen by easily recognisable shapes. An action can be selected by moving the cursor over the relevant icon, and then pressing one of the selection buttons, which immediately changes the cursor into the shape of the selected icon. The picture can then be built up by moving the cursor over the drawing area and, by using the selection buttons, you can perform any of the drawing functions. Because the movement of the cursor so exactly replicates the motion of the mouse, and the use of the program is so straightforward, the excellent manual provided by AMX Art is hardly necessary.

The lower menu at the side of the screen selects the line widths and shapes in the same manner. Across the bottom of the screen is a third menu for selecting the shading patterns for filling in shapes — ranging from plain black to complex trellis patterns. Because the display is two colours only — due to memory constraints — these patterns are provided to distinguish different areas with different types of shading.

Other options, such as SAVEing or LOADing pictures, operating system commands, and actuating the built-in printer dump program, can be selected via the 'pull-down' menus at the top of the screen. Only the titles of these menus are displayed, and by positioning the cursor over one of these and pressing a selection button, you can reveal the entire menu over a section of your picture. After selecting your options with the mouse, the menu vanishes, leaving the picture in its original form.

Complex, accurate and artistically innovative compositions can be simply and quickly rendered when you first use the program. Using this package after a very short time becomes as comfortable as using pen and paper, except, of course, that AMX Art will neither blunt nor smudge, and it can be reproduced as many times as you want.

The Art program is reason enough to buy the mouse package, but AMX intends to expand these beginnings into an entire mouse-based system for the BBC Micro. A desk-top manager, similar in operation to that used on the Macintosh and Lisa computers, is in the works and will sell for about £25. Also planned is a mouse-driven database and further enhancements to the AMX Art program, the latter incorporating colour and a zoom facility. The mouse and pull-down menus look to be a sure addition to the popular Wordwise word processor program.

The AMX mouse and the AMX Art program, as well as the other forthcoming mouse- and icon-based programs, look set to change the face of the BBC Micro for the user wishing to expand the machine's capabilities, and with their ease of use, these programs have set a new standard for home computers.

AMX MOUSE
PRICE
£80
DIMENSIONS
85×65×35 mm
INTERFACES
A cable to plug into BBC Micro's user port
DOCUMENTATION
2 manuals that are both easily understood and informative, though hardly needed for the Art package because of the program's user-friendliness
STRENGTHS
Simple to operate, excellent manuals, Art package has many capabilities and is adapted well to the Micro's operating system
WEAKNESSES
Apart from the Art package, the software is somewhat limited. The metal ball slips on hard shiny surfaces

ALL ABOARD!

Having looked at the general principles involved in simulations (see page 1026), we now embark on our project to create a game based on a 16th century trading expedition to the New World. Written in Microsoft BASIC, it is designed so that the listings given in each instalment can be run as independent modules.

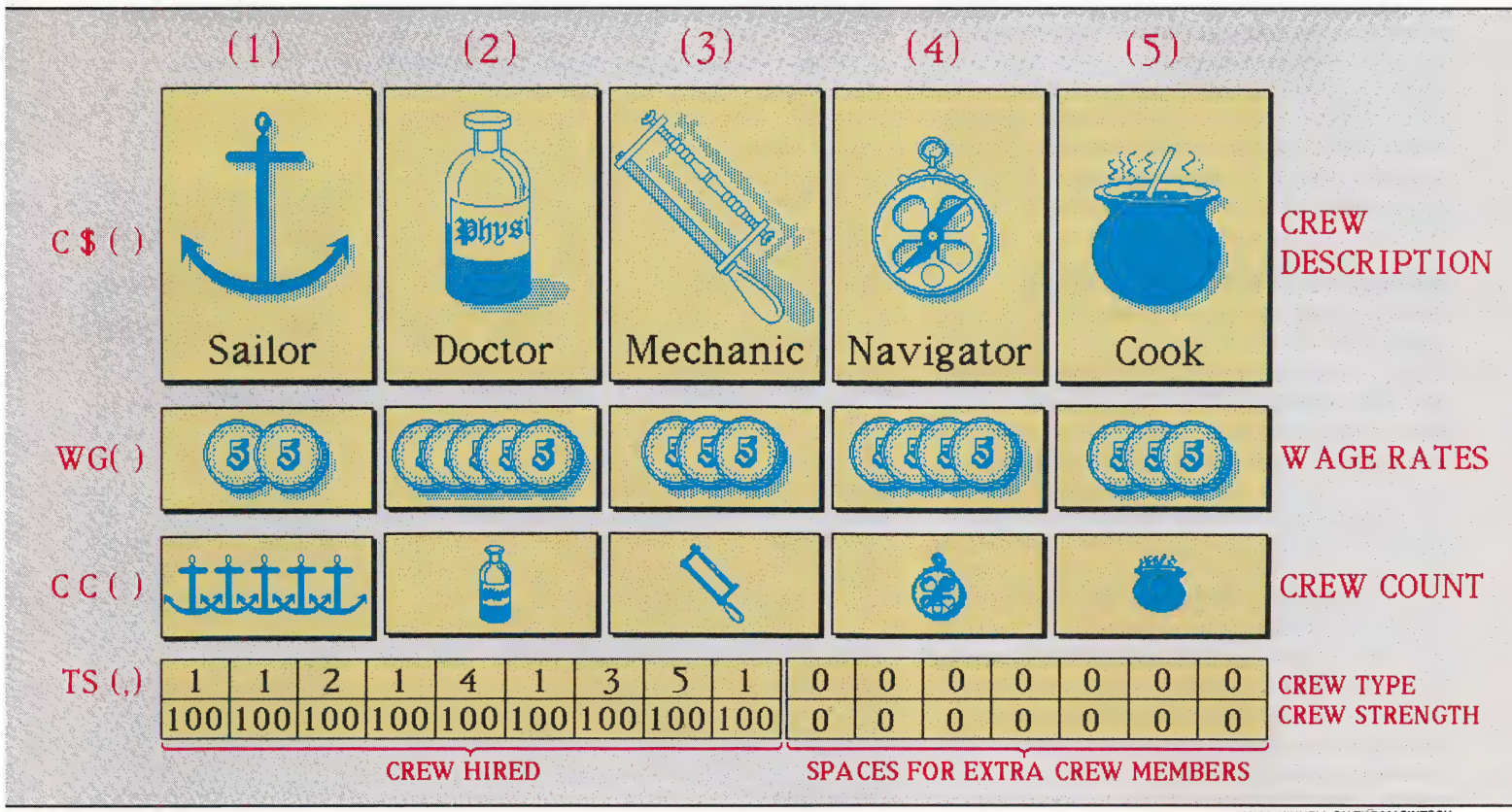
In this simulation game you are taken back through time and given the role of a 16th century trader, having to organise an expedition to the New World. You must plan the voyage, make the journey and finally trade (profitably, you hope) with the inhabitants of the New World.

The game begins with the player owning a trading ship and having 2,000 gold pieces to spend. Your first task is to hire a crew, offering each a weekly wage during the expedition; the level of payment will depend on the skills the crew member has. The crew must comprise a range of skills, whilst not overburdening yourself with an excessive salary budget. The trading vessel will accommodate sixteen crew members — larger crews will of course cost more and consume larger amounts of food and drink, leaving less money for trading. However, if the crew is too small then it may not be able to cope with some incidents during the voyage, such as scurvy or pirate attacks, and the ship will not run as efficiently, making the journey longer and costing more gold.

The first game module is concerned with initialising variables and arrays that will be used during the game and for hiring members of the crew. The first section of the module DIMensions arrays that hold important information about the ship's crew. C$() is used to hold descriptions of the crew types. As there are five categories of crew, the array is DIMensioned to have five elements in line 16. The actual descriptions are then assigned individually to each array element. It is much more convenient — and it saves memory — to refer to the crew descriptions by their element number in this array. This element number is sometimes known as the array address because it gives the position of the description in the array.

The status of the crew varies throughout the game, their strength increasing and decreasing in the light of circumstance. To keep a record of the crew's makeup and the strength of each individual crew member, a two-dimensional array TS(,) is used. The array is DIMensioned to have 16 columns and two rows. Each column represents a

Press Gang
Four arrays are used to hold data about the crew hired for the voyage. C$(), WG() and CC() hold the characteristics associated with each crew type: descriptions, wage rates and the number hired. TS(,) holds the actual composition of the crew selected for the voyage, up to a maximum of 16 members. Each crew type can be referred to by a number rather than a name — the number corresponding to an address in the three crew characteristics arrays



crew member, the upper row containing the crew type and the lower row containing the strength rating. Initially, each crew member has a strength rating of 100.

Wage rates for each crew type can also be held as a five element array WG(). Thus if we want to know the wage rate of a doctor, crew type 2, we can look it up in this array as WG(2). Sailors receive 10 gold pieces a week, doctors get 25, navigators receive 20 gold pieces, and cooks get 15 each. To keep track of the amount of money possessed by the player, a variable MO is initialised to 2000 in line 12 and will be decreased as weekly wage bills are met and trading begins.

Later in the game the program will need to know how many of each crew type have been hired so that it can calculate the efficiency of the crew in dealing with contingencies that may arise. The makeup of the crew could be determined whenever required by scanning through the upper row of the array TS, but it is quicker and easier to set up another array CC() to hold the number of each type in the hired crew. Again this array only requires five elements, and this is DIMensioned in line 18.

INTRODUCTORY INFORMATION

Having set up these arrays to hold data about the crew, the program module then PRINTs information about the approximate length of voyage and about money that must be set aside for provisions. This information is not PRINTed in its entirety but a special routine is used to make the letters appear slowly across the screen. This subroutine, at line 9100, accepts the phrase to be printed in the variable S$ and PRINTs one character at a time to the screen, inserting a short delay to achieve the teletype effect. A second subroutine, at line 9200, inserts longer pauses, when required, between one line of text appearing and the next.

The subroutine at line 1000 allows you to select your crew. This is done by selecting a crew type 1 to 5, or pressing F to finish. Valid entries are stored in TS(,) at line 1156, with CN keeping a count of the number of crew hired so far. As each crew member is hired, the weekly total wage bill WT is displayed, having been calculated by adding the relevant element of the array WG() to the previous value of WT. This happens at line 1158. At each stage the current composition of the crew is PRINTed out by the section of program between lines 1160 and 1190. Line 1185 checks if the crew count for a particular type is greater than one or zero. If this is the case then an S is added to the end of the crew type description.

At several stages of the program the player is asked to press a key before progressing. The variable K$ is used to hold the message 'PRESS ANY KEY TO CONTINUE' and is transferred to S$ for PRINTing whenever the message is required. On returning from the crew hiring subroutine, the first module ends. In the next instalment of our programming project, we shall be looking at a module that allows you to buy provisions.

Module One

```
9 PK$="  PRESS ANY KEY TO CONTINUE"
10 DIM TS(16,2): REM CREW TYPE/STRENGTH
11 CN=0: REM NO. OF CREW
12 MO=2000: REM START MONEY
13 DIMWG(5):WG(1)=10:WG(2)=25:WG(3)=15:WG(4)=20:WG(5)=15: REM WAGES
14 WT=0: REM WEEKLY WAGE BILL
15 :CM=16:REM MAX CREW
16 DIMCD$(5):CD$(1)="SAILOR":CD$(2)="DOCTOR":CD$(3)="MECHANIC"
17 CD$(4)="NAVIGATOR":CD$(5)="COOK"
18 DIMCC(5): REM COUNT OF EACH CREW TYPE
80 PRINTCHR$(147):ST$="  NEW WORLD TRADING GAME*":GOSUB9100:PRINT
81 GOSUB9200
82 ST$="YOU ARE THE CAPTAIN OF A SHIP*" :GOSUB9100:PRINT
83 ST$="SAILING TO THE NEW WORLD. THE*":GOSUB9100:PRINT
84 ST$="JOURNEY IS SAID TO TAKE EIGHT*":GOSUB9100:PRINT
85 ST$="WEEKS,BUT MAY TAKE LONGER. YOU*":GOSUB9100:PRINT
86 ST$="MUST HIRE A CREW,PAY THEM, BUY*":GOSUB9100:PRINT
87 ST$="PROVISIONS,EQUIPMENT AND GOODS*":GOSUB9100:PRINT
88 ST$="FOR TRADING. YOU HAVE 2000 GOLD*":GOSUB9100:PRINT
89 ST$="PIECES TO SPEND.*":GOSUB9100:PRINT:GOSUB9200
90 PRINT:ST$="          GOOD LUCK!*":GOSUB9100:GOSUB9200:PRINT
91 ST$=PK$:GOSUB9100
94 GETIP$:IFIP$=""THEN94
95 GOSUB9200
500 GOSUB1000
999 END
1000 PRINTCHR$(147): PRINT" STAGE 1 - HIRING CREW"
1010 PRINT" ----------------------"
1012 PRINT
1015 GOSUB9200
1020 PRINT:PRINT"CREW TYPES AVAILABLE:"
1025 GOSUB9200
1030 PRINT
1040 PRINT"TYPE DESCRIPTION WAGES PER WEEK"
1050 PRINT"---- ----------- ---------------"
1060 PRINT"   1   SAILOR        10 GOLD PCS"
1070 PRINT"   2   DOCTOR        25 GOLD PCS"
1080 PRINT"   3   MECHANIC      15 GOLD PCS"
1090 PRINT"   4   NAVIGATOR     20 GOLD PCS"
1100 PRINT"   5   COOK          15 GOLD PCS"
1105 GOSUB9200
1110 PRINT:PRINT:PRINT
1120 ST$="ENTER CREW TYPE REQUIRED(1-5)*":GOSUB9100
1122 ST$="OR 'F' TO FINISH HIRING*":GOSUB9100:PRINT:INPUTIP$
1125 CT=VAL(IP$)
1128 IFLEFT$(IP$,1)="F"THENPRINT:PRINT"END OF CREW HIRE.":GOSUB9200
     :GOTO13:D
1130 IFCT>0ANDCT<6THEN1150
1139 PRINT:PRINT
1140 PRINTIP$;:ST$=" IS NOT A CREW TYPE*":GOSUB9100
1142 GOSUB9200
1145 ST$="PLEASE ENTER AGAIN"
1146 GOSUB9100
1147 GOTO1300
1150 PRINT:PRINT
1155 CN=CN+1:REM CREW HIRED SO FAR
1156 TS(CN,1)=CT:REM CREW TYPE
1157 TS(CN,2)=100:REM STARTING STRENGTH
1158 WT=WT+WG(CT):REM TOTAL WAGES
1159 CC(CT)=CC(CT)+1:REM CREW TYPE COUNT
1160 ST$="CREW SO FAR:"
1170 FORT=1TO5
1180 PRINTST$;CC(T);" ";CD$(T);
1185 IFCC(T)>1ORCC(T)=0THENPRINT"S":GOTO1189
1186 PRINT" "
1189 ST$="                "
1190 NEXT
1195 PRINT: PRINT"TOTAL WEEKLY WAGE BILL ";WT
1200 IFCN=CM-1THENPRINT:ST$="ONLY ONE MORE CREW*":GOSUB9100:GOTO1295
1202 IFCN=CMTHENPRINT:ST$="    SHIP NOW FULL!!":GOSUB9100:GOTO1310
1295 REM
1300 GOTO1015
1310 PRINT: ST$=PK$:GOSUB9100:PRINT:   GOSUB9200
1320 GETIP$:IFIP$=""THEN1320
1998 RETURN
9100 REM SLOW PRINT OF ST$
9110 FOR S3=1TO32
9115 IFMID$(ST$,S3,1)="*"THENS3=32:GOTO9140
9120 PRINTMID$(ST$,S3,1);
9130 FORS4=1TO25:NEXT
9140 NEXT:PRINT
9199 RETURN
9200 REM DELAY LOOP
9210 FORS5=1TO1000:NEXT
9299 RETURN
```

Basic Flavours

Spectrum:

Make the following changes:

```
16 DIM C$(5,9):C$(1)="SAILOR":C$(2)=
   "DOCTOR":C$(3) ="MECHANIC"
80 CLS:LET S$="NEW WORLD TRADING
   GAME*":GOSUB 9100: PRINT
94 LET P$=INKEY$:IF P$="" THEN GO TO 94
1005 CLS:PRINT" STAGE 1 — HIRING CREW"
1128 IF P$(1 TO 1)="F" THEN PRINT"END OF
   CREW HIRE."GOSUB 9200:GOTO1310
9115 IF S$(S3 TO S3)="*" THEN S3=32:
   GOTO 9140
9120 PRINT S$(S3 TO S3);
```

BBC Micro:

Make the following changes:

```
80 CLS:S$="NEW WORLD TRADING
   GAME*":GOSUB 9100:PRINT
94 P$=GET$
1005 CLS:PRINT" STAGE 1 — HIRING CREW"
```

Making Preparations . . .

The first module of the New World trading game is concerned with hiring crew. Several arrays are set up for this purpose and the player can select up to 16 crew members as accompaniment on the expedition

SCHOOL TURTLES

The combination of the BBC Micro and LOGO is rapidly becoming the preferred school-based computer system. With four versions of LOGO now available, we have chosen the two mostly likely candidates for the classroom — packages marketed by Acornsoft and Logotron — and examine their potential as viable educational tools. These two versions are both full LOGO implementations using the LCSI syntax, so if you followed our LOGO course, you will have no difficulty getting used to either version.

The graphics screen is about 1200 by 800 turtle steps, giving better resolution than LOGO packages for most other micros. All the BBC Micro's eight screen modes are available, as are the various drawing options (dotted lines, triangular fill, etc). If you use Mode 7, you release a lot more memory for applications that do not require graphics. For sound, the SOUND and ENVELOPE commands are provided. These work in exactly the same way as they do in BBC BASIC.

You can use COPY to edit lines in the usual way, and you can use the BBC Micro's control codes, as well as VDU, *FX and other * commands. Both versions claim to support the second (6502) processor and Econet. They both worked fine with Econet, but we did not try them with the second processor.

Written in BCPL and then compiled, the Acornsoft version is provided on two 16 Kbyte ROM's, and is accompanied by three books, a tape and a disk of sample programs and LOGO extensions. It is packed with features, having well over 200 primitives, including those in the extensions on disk. The choice of primitives seems to have been dictated by an attitude to include as much as possible.

"Much early work in artificial intelligence used LOGO, and it is closely related to the leading language for designing expert systems, LISP."

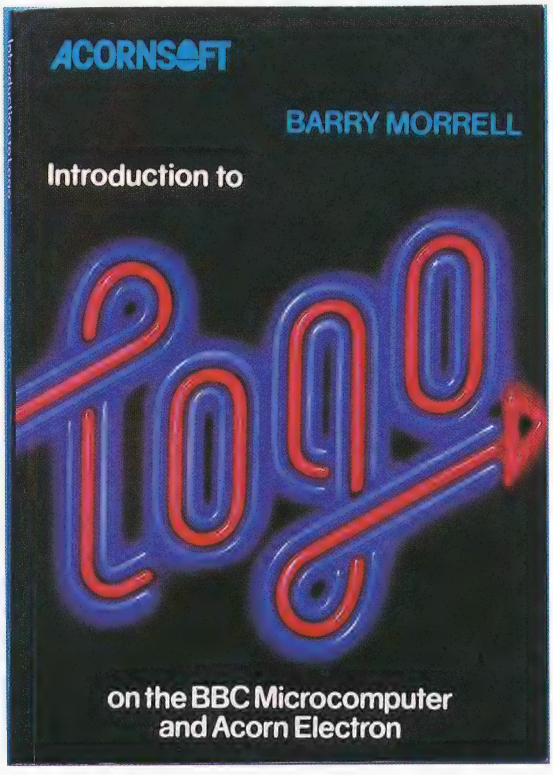
Let's look at the most important of the extra features. One set of primitives makes it possible to set up modified LOGO environments, tailoring them for student use. Two features are particularly useful; primitives can be redefined, and procedures can be 'buried' so that they appear to be primitives; and files can be set up to run an initialisation procedure on loading.

To help in the debugging of programs, there are a wide variety of trace options — every line, every procedure call, or every primitive, with the option of pausing at each point, can be traced. This is an excellent new feature that also helps explain to a student the logic of a program. CATCH and THROW provide a way of jumping out of one procedure into another. These primitives are used mainly for trapping input errors, and there are a number of other primitives provided to help you with this.

"LOGO has two types of object, 'words' and 'lists'. It also has operations which allow you to join objects together, break them into distinct parts, or examine them."

The extensions provided on the disk offer a number of other features. There is a set of primitives for dealing with 'properties' (Apple LCSI LOGO also has these), the idea of which is derived from LISP. The creation of multiple turtles (up to 32), each capable of having a unique shape and being addressed individually, is possible with yet another set of primitives. There are extensions

Acornsoft LOGO
by Chris Jobson and John Richards; produced by Acornsoft Ltd, Betjeman House, 104 Hills Road, Cambridge CB2 1LQ; ROM-based; £69



that enable turtle commands to be given to the BBC Buggy, the Jessop turtle or the Valiant turtle. We tried out the Buggy extension and found it very easy to program.

The Logotron version was written in machine code and it comes on a single 16 Kbyte ROM together with a ring-bound tutorial and reference manual. It was developed by SOLI (Systèmes d'Ordinateur Logo International), who have worked closely with LCSI in the past, and were also responsible for the Spectrum LOGO. This version is very much like any other LCSI LOGO.

With about 120 primitives, SOLI has made a deliberate choice to restrict the number of available facilities in order to fit them onto a single ROM. This saves on the use of ROM slots, and also allows greater speed. The Logotron version has a number of extra features worth noting, one of which allows programs to be written to and read from the supported text files, as in Atari LOGO. Great care has been taken over the design of the screen editor. The function keys have been used to control the editor, and FIND and REPLACE commands have been incorporated.

"The most important feature of LOGO is that you can make it reflect your needs, interests, and personality."

"LOGO is widely regarded as a 'programming language for children'. It also happens to be a 'programming language for computer scientists'."

Two new primitives are provided, OPPS and OPNS, which output lists of procedure names and variable names, respectively. These can be used to provide alternatives to the more usual POTS and PONS commands. A very useful feature allows variables to be passed to BBC OS commands. This lets you write coherently named procedures to replace the cryptic messages normally required by the BBC OS.

Two primitives are noticeably absent: TEXT and DEFINE. Definitions for these are given in the manual, but since they operate by writing to a file and reading it back again, they are obviously going to be slow. Structured programmers will be pleased to note that, as with LOGO on the Spectrum and Atari, lack of space has prevented the inclusion of a GO statement. The TRACE command prints the name of each procedure, and its inputs, as it is entered, but there is no message on leaving it. A fuller TRACE option would have been very useful.

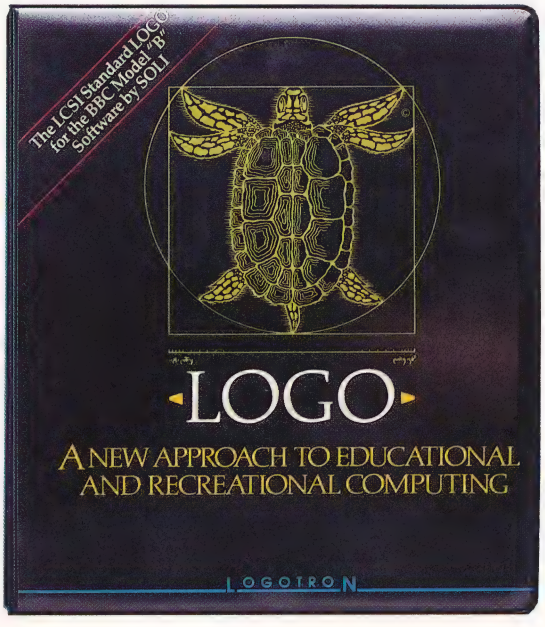
We ran a number of benchmark programs in order to give some comparisons between the two versions in terms of workspace and speed. The LCSI version has a little more space for defining variables and procedures, and about twice as much 'stack' space, thus allowing more levels of recursion before running out of memory. Both versions implement 'end recursion' efficiently for commands and operations. This means that end recursive procedures go on indefinitely.

On various tests of speed, the Acornsoft version took from 50 to 1,000 per cent longer than the Logotron version, with the most significant differences being found in list processing and assignments. (The garbage collector on the Acornsoft version seemed to be particularly slow.) This slowness in list processing with the Acornsoft version could be very irritating; however, a PROLOG-like logic language written in LOGO, which is supplied on the disk, ran with acceptable speed.

Both versions come with detailed manuals, which include tutorial sections for beginners and reference sections. These give comprehensive explanations of turtle geometry, although the Acornsoft version consistently obscures the command/operation distinction. Each manual devotes just eight pages to describing list processing, but the Logotron version makes up for this by including the listings of some procedures that could make list processing much easier for students to learn. (These procedures are the results of work done by Mike Sharples at the Edinburgh University AI Dept.)

Acornsoft provides a number of lengthy example programs on the disk (or cassette), including a PROLOG-type program, and an example of 'natural language' parsing.

Acornsoft is promising further extensions in the future, but no details have been given. Logotron expects to market an 'Advanced LOGO' soon, on disk, which will add many more primitives to the language. More exciting is their promise of a sprite board, featuring 30 'hard' sprites. Both companies are offering discounts for large orders, which should be most attractive to schools. Logotron claims its software is uncopyable and offers to replace damaged ROM's for £25.



IAN McKINNELL

Logotron LOGO
by Gerard Dehan, Armen Valian, Marie Paule Deprund and Eric Millet; produced by Logotron Ltd, Ryman House, 59 Markham Street, London SW3; ROM-based; £69.95

BASIC INTERACTION

The previous instalment's discussion of the nature of BBC BASIC serves as a prelude to our look, here, at the way BASIC interacts with the BBC Micro's operating system.

We discussed, at the beginning of this series on microcomputer operating systems (see page 858), how an OS gives us protection against any changes in the hardware configurations of computers. It is not surprising to find, therefore, that BASIC uses OS calls extensively. The BASIC SOUND command, for example, generates a call to OSWORD with A=7 (see page 978) to produce the sound; similarly, ENVELOPE makes use of OSWORD with A=8. The interesting thing to note here is that we can modify the way in which BASIC behaves to a certain extent by altering the way in which the operating system behaves — i.e. by altering vectors and writing our own sections of code to replace OS routines.

Similarly, OSWRCH, OSASCII and OSNEWL are used by the various routines that access the screen. One point to note is that machine code routines that produce graphics displays using the OS routines are not always much faster than the equivalent BASIC commands. This is a measure of the efficiency of the BASIC interpreter's use of the OS calls when interpreting such calls as MOVE, DRAW and PLOT.

Let's look at the two BASIC commands that are used to call machine code programs from BASIC, be they OS routines or our own programs. The instructions are USR and CALL.

THE USR CALL

The role of the USR instruction is to run a piece of machine code and return a value to BASIC that gives us details of the state of the A, X and Y registers after the code has been executed. Information is also returned concerning the status of the carry flag and the processor status register (PSR). It is called in the following way:

```
result%=USR(address%)
```

where address% is the address of the routine that is to be executed. Before you issue this call, you can put values into the A%, X% and Y% variables that will be passed by the USR routine into the A, X and Y registers, respectively. The least significant bit of the C% variable can also be used to affect the value of the carry flag; an even C% will set it to zero.

The following program shows a short example of the use of USR; here we're calling one of the OSBYTE calls that returns a value in the X and Y registers. This routine returns the x co-ordinate of the text cursor in the X register, and the y co-ordinate of the text cursor in the Y register. The value in result% represents these registers in a coded form, which we'll discuss a little later.

```
10 A%=134
20 X%=0
30 Y%=0
40 LET result%=USR(&FFF4)
50 PRINT result %:REM prints the hex representation
```

Printing the result in hexadecimal makes it easier for us to decode the values returned from the OSBYTE call.

USR Function In Action

The USR function enables the user to call from BASIC a section of machine code designed to return a value to the BASIC program. Because USR is a function (as opposed to CALL, which is a statement) and returns a value, the result must be assigned within the BASIC program, either to a variable:

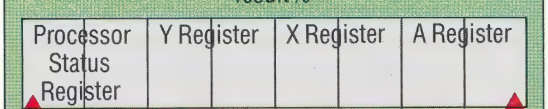
```
result%=USR(&FFF4)
```

or as part of a PRINT statement:

```
PRINT USR (&FFF4)
```

The result returned to BASIC by the USR function takes the form of a four-byte integer number that reflects the contents of the four registers — P, Y, X and A — as follows:

result%=



Most Significant Byte — Least Significant Byte

We can get the desired value for a given register by using the AND operator to mask off certain parts of the result. For example:

```
result%AND&000000FF
```

will mask off everything except the value of the A register

THE CALL COMMAND

The other method of calling machine code programs is the CALL command. We've already used this several times — in particular, for calling OSWORD and the other OS routines. In that case, we simply called the routine by specifying it's address. For example:

```
CALL &FFF1
```

BBC BASIC also allows us an extended version of this command, which enables the values held in

BASIC variables to be available to a machine code program. Obviously, the main problem with such an arrangement is telling the machine code program where it can find the variables in memory. Fortunately, this problem is solved by the BASIC interpreter; on making a CALL with parameters, such as:

```
CALL  address%,A%,C%
```

all the details about the parameters — in this case A% and C% — are stored in page 6 of the memory. This area of memory is known as a 'parameter block', and the following diagram shows how such a block is arranged.

Location	Description Of Contents
&0600	Number of parameters
&0601	Address of first parameter
&0602	
&0603	Type of first parameter
&0604	Address of second parameter
&0605	
&0606	Type of second parameter

Obviously, the address and type entries are made as many times as needed. The address of the parameter is fairly straightforward; it's where in memory we'll find the first byte of the parameter. The type entry indicates what type of variable the parameter is. The following table shows the range of parameter types available:

Type	Description	Example
0	8-bit byte	CALL address, ?&70
4	Integer variable	CALL address, F%
8	Real variable	CALL address, G
128	String at an address	CALL address, $(&A00)
129	String variable	CALL address, F$

The final entry, parameter type 129, is slightly different: it's not quite true that the address entry in the parameter block points to the position of the string in memory. It points, in fact, to an area of memory that provides us with more details about the string. This area of memory is called a 'string information block', and is arranged in this format:

Location	Description Of Contents
Address held in parameter block	Address of string
	Bytes allocated
	String length

Thus, the first two bytes of the information block point to the first character in the string. The 'bytes allocated' entry will show the maximum length to which the string has been assigned before it's current length, which is in the fourth byte.

PARAMETER PASSING

The CALL statement is used to execute a section of machine code from BASIC, but, unlike the USR function, does not return a result. The user can, however, include a subroutine that maps values from the machine code program onto BASIC variables, enabling them to be read on return.

The following program not only gives an example of parameter passing using the CALL statement, but also maps a result onto the integer variable A%. Further, it demonstrates how we can generate errors from within machine code programs that make use of BASIC error handling (see page 1038). It's written for BASIC II.

```
 10  DIM C 400
 20  zero=&70
 30  FOR I%=0 TO 2 STEP 2
 40  P%=C
 50  [ OPT I%
 60  .code LDA &600
 62  CMP #1
 64  BNE error
 70  LDA &603
 80  CMP #4
 90  BNE error2
100  LDA &601:STA &70
110  LDA &602:STA &71
120  LDY #0
130  LDA &02:STA (zero),Y:INY
140  LDA &03:STA (zero),Y
160  RTS
170  .error BRK
180  EQUB 254
190  EQUS "Not correct number of parameters"
200  EQUB 00
210  .error2 BRK
220  EQUB 253
230  EQUS "Type Mismatch"
240  EQUB 00
250  ]:NEXT
260  A%=0:CALL code,A%
270  PRINT"End of BASIC variables is ";A%
```

The program returns the address of the end of BASIC variables in the integer variable passed to it as a parameter — thus demonstrating that CALL can also pass values back to BASIC. Lines 60 to 64 check for only one parameter, and should this condition not be satisfied an appropriate error message is given. Lines 70 to 90 then check that the parameter that has been passed to the routine is an integer type, and an error is generated if this is not so. Lines 100 to 160 transfer the data from addresses 2 and 3 to the least significant bytes of the integer variable that was passed as a parameter. Lines 170 to 240 are used to generate the error messages.

The CALL is made in line 260; A% is set to zero before the CALL is made because the machine code program doesn't alter the most significant bytes of the integer variable involved. After the call, the expression HIMEM—A% will give you the number of bytes left when the BASIC program and variables have taken their share of the memory.

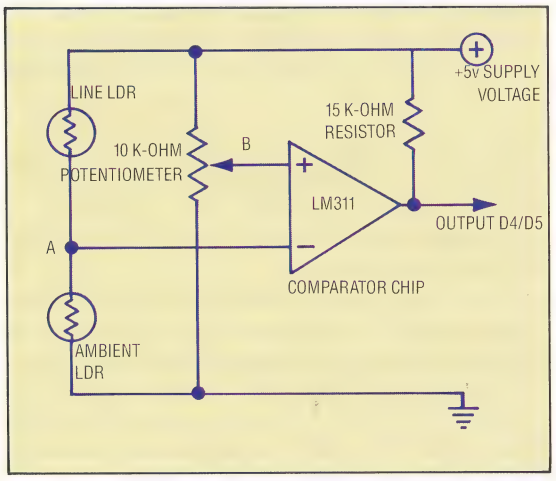
VISUAL AWARENESS

By equipping our robot with a bank of light-dependent resistors, we give it sufficient 'sight' to follow a dark line. Programmer control via the BBC Micro, Commodore 64 or Workshop-converted Spectrum is made possible by data feedback from these light sensors.

The unit that enables the robot to follow a line consists of two identical circuits, each feeding a different input line of the user port. Each circuit is based around an LM311 comparator chip, which compares the voltages at its two input pins. If the voltage at the positive pin is higher than that at the negative input, the output will swing from zero volts to the supply voltage. In logic terms, the output from this chip will be zero if the two input voltages are the same and one if the voltage at the positive input pin rises above that at the negative input pin.

Two light-dependent resistors (LDRs) on each circuit form a voltage divider between the supply voltage and earth. If the light falling on the two LDRs is the same then their resistance will also be equal and the voltage at point A in the circuit diagram will be about 2.5v (half the supply voltage). The present potentiometer can be adjusted to give the same voltage at point B. As the two input voltages are the same the output from the comparator chip will be zero. If the light falling on the line LDR decreases — for example, when it is over a dark line — its resistance will rise, causing the voltage at point A to go above 2.5v. This in turn will cause the comparator output to swing to the supply voltage, giving a logic output of one indicating that the line LDR is over the dark line.

The presence of two circuits, each consisting of a pair of LDRs and a comparator chip, enables the computer to detect which side of the line the robot has drifted. In each pair of LDRs, one measures the ambient light intensity so that it can be compared with the light intensity measured by the other LDR over the line.



Eyes Down
The line follower circuit is made up of two identical circuits, each using a pair of light-dependent resistors. A comparator chip is used to produce a digital signal if there are differing amounts of light falling on the LDRs. In this way the robot can detect whether it is over a blank line or not

CALIBRATION

After constructing and fitting the board we must calibrate it. In lighting conditions similar to those in which you wish to run the line follower program, run this calibration program and adjust the two preset potentiometers on the board just installed.

LDR Test Programs

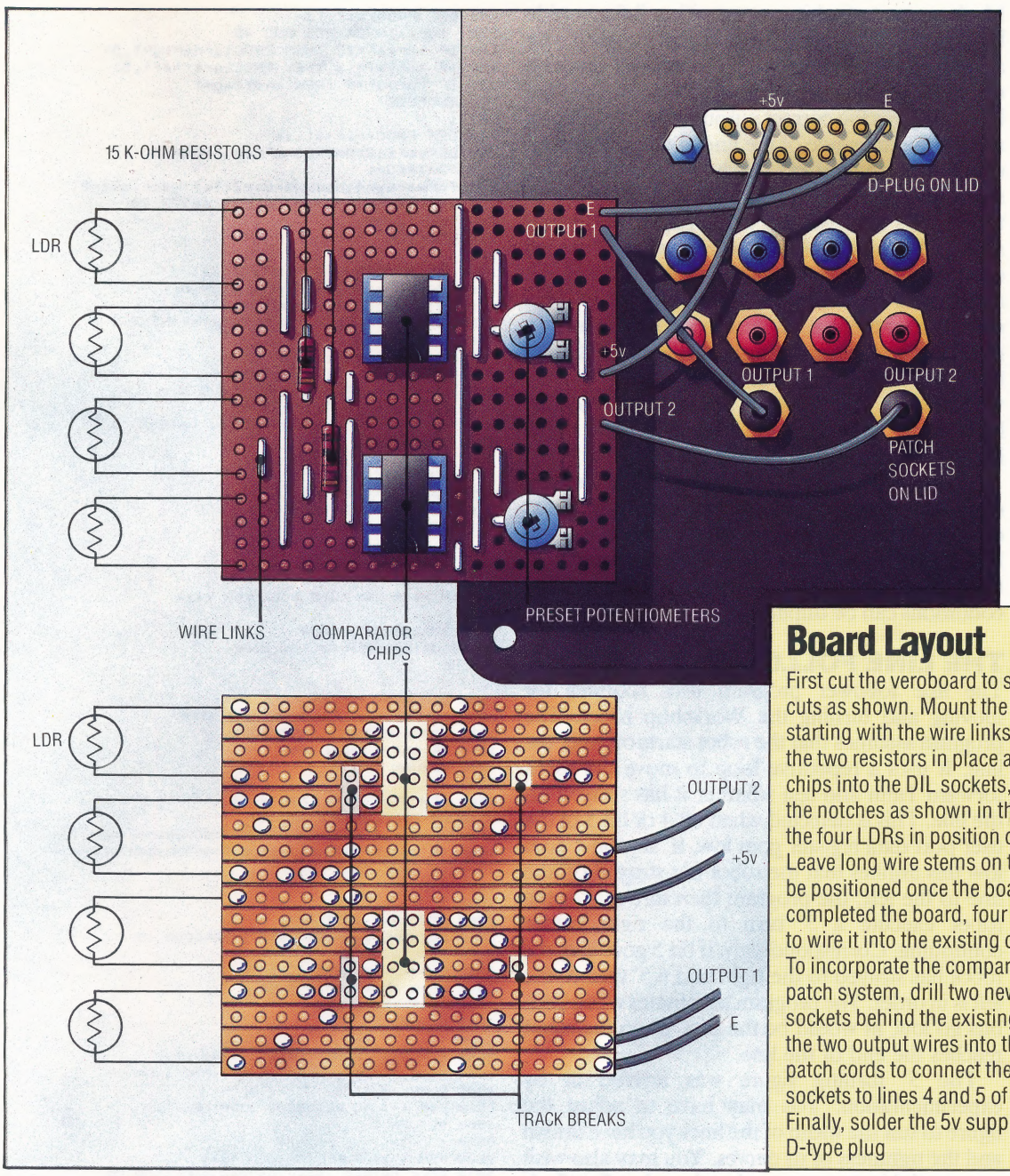
```
1000 REM **** BBC LDR TEST ****
1010 DDR=&FE62:DATREG=&FE60:?DDR=15:REM LINES
     0-3 OUTPUT
1020 REPEAT
1030 contents=?DATREG
1040 IF(contents AND 16)=0 THEN PRINT TAB(5)
     "LEFT";
1050 IF(contents AND 32)=0 THEN PRINT TAB(15)
     "RIGHT";
1060 PRINT
1070 UNTIL FALSE
```

```
1000 REM **** CBM LDR TEST ****
1010 DDR=56579:DATREG=56577:POKE DATREG,15
1020 CN=PEEK(DATREG)
1030 IF(CN AND 16)=0 THEN PRINTTAB(5)"LEFT";
1040 IF(CN AND 32)=0 THEN PRINTTAB(15)"RIGHT";
1050 PRINT
1060 GOTO 1020
```

```
1000 REM **** SPECTRUM LDR TEST ****
1010 CLEAR 32499:LET ST=32500: GO SUB 3000
1020 LET NM=16:GO SUB 2000:IF USR ST=0 THEN PRINT
     TAB 5;"LEFT";
1030 LET NM=32:GO SUB 2000:IF USR ST=0 THEN PRINT
     TAB 5;"RIGHT";
1040 PRINT
1050 GO TO 1020
1060 :
2000 REM **** PERFORM AND ****
2010 POKE ST+1,IN 31
2020 POKE ST+3,NM:RETURN
2030 :
3000 REM **** MACHINE CODE LOADER ****
3010 FOR I=ST TO ST+8
3020 READ A:POKE I,A
3030 NEXT I
3040 DATA 62,0,14,0,161,6,0,79,201
3050 RETURN
```

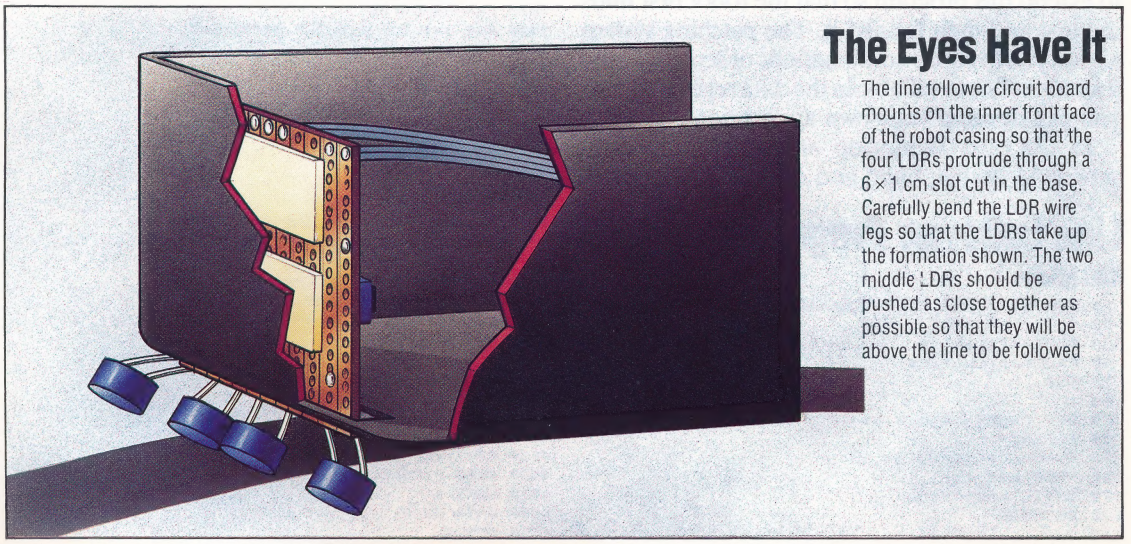
Parts List

No	Item	Maplin Number
2	LM311 comparator chips	QY09K
2	8-pin DIL sockets	BL17T
2	2mm sockets	HF44X
2	15 k-ohm 0.4 watt resistors	M15K
2	10K horizontal preset potentiometers	WR58N
4	ORP12 LDRs	HB10L
1	50-hole by 24-strip veroboard	FL07H
1	Strip self-adhesive pads	HB22Y
1m	4-way ribbon cable	—
1m	Bare 20 swg tinned wire	—



Board Layout

First cut the veroboard to size and make the track cuts as shown. Mount the passive components first, starting with the wire links and DIL sockets. Solder the two resistors in place and insert the comparator chips into the DIL sockets, noting the orientation of the notches as shown in the diagram. Finally, solder the four LDRs in position on the edge of the board. Leave long wire stems on the LDRs to allow them to be positioned once the board is in place. Having completed the board, four leads must be connected to wire it into the existing circuitry in the robot's lid. To incorporate the comparator chip outputs into the patch system, drill two new holes and mount the sockets behind the existing group of eight. Solder the two output wires into these sockets. Use two patch cords to connect the line follower board output sockets to lines 4 and 5 of the patch socket group. Finally, solder the 5v supply and earth wires to the D-type plug



KEVIN JONES

The Eyes Have It

The line follower circuit board mounts on the inner front face of the robot casing so that the four LDRs protrude through a 6 × 1 cm slot cut in the base. Carefully bend the LDR wire legs so that the LDRs take up the formation shown. The two middle LDRs should be pushed as close together as possible so that they will be above the line to be followed

Mark a piece of white paper with a 2.5 cm wide black line. Taking each preset in turn, adjust it with a small screwdriver so that the computer gives no output when the two central LDRs are over the line, but gives the correct output when the robot is moved to either side of the line. After adjustment check the function by slowly moving the robot sideways across the line from left to right. While all four LDRs are over the white paper, the screen should display LEFT RIGHT. As the robot is moved over the line, the message should change to LEFT, and then to no message as the robot is positioned directly over the line. As the robot is shifted further right, the screen should display RIGHT and finally LEFT RIGHT when all four LDRs are again over white paper. The program assumes that the left-hand LDR is patched into line 4 and the right-hand LDR is patched into line 5.

The operating light conditions must remain constant during calibration and on subsequent occasions when the line follower program is used. It is a good idea to floodlight the operating area using a bright overhead light, so that lighting conditions can be duplicated on future occasions.

THE LINE FOLLOWER PROGRAM

The line follower program uses routines for moving and turning the Workshop robot. The program assumes that the robot starts off over the line. It uses a repetitive loop to move the robot forward 1mm and test whether it has strayed off the line. This is detected when bit 4 or bit 5 of the user port data register goes low. If, say, bit 4 goes low this implies that the robot has strayed off the line to the left; the program therefore makes the robot execute a 5° turn to the right before continuing. Correspondingly, if bit 5 goes low, the robot has strayed to the right and a 5° turn to the left is in order. The program terminates when both bits go low, as would be the case when the robot reaches the end of the line.

The 5° turning figure was arrived at by experimentation. You may have to adjust the figure to suit the width of the lines you have drawn and the tightness of its curves. You may also wish to extend this program so that the robot first finds the line and then follows it. The patching system allows you to set up combinations of sensors to the four input lines available in the data register so that you can develop your own applications.

In the next Workshop article we start on a project to design, build and control a robot arm.

Line Follower Programs

BBC Micro

```
 10 REM **** BBC LINE FOLLOWER ****
 20 :
 30 PROCinitialise
 40 PROCfollow_line
 50 END
 60 :
 70 DEF PROCfollow_line
 80 REPEAT
 90 PROCmove(forwards,1)
100 PROCtest_ldr
110 UNTIL endflag=1
120 ENDPROC
130 :
140 DEF PROCtest_ldr
150  ldrval=?DATREG AND 48
160 IF ldrval=32 THEN PROCturn(right,5)
170 IF ldrval=16 THEN PROCturn(left,5)
180 IF ldrval=0 THEN endflag=1
190 ENDPROC
200 :
210 DEF PROCinitialise
220 DDR=&FE62:DATREG=&FE60:?DDR=15
230 ?DATREG=1
240 forwards=4:backwards=2:left=6:right=0
250 pd_ratio=3.34446:pa_ratio=379/90
260 endflag=0
270 ENDPROC
280 :
290 DEF PROCmove(dir,distance)
300 ?DATREG=(?DATREG AND 1)OR dir
310 pulses=pd_ratio*ABS(distance)
320 FOR I=1 TO pulses:PROCpulse:NEXT I
330 ENDPROC
340 :
350 DEF PROCturn(dir,angle)
360 ?DATREG=(?DATREG AND 1)OR dir
370 pulses=pa_ratio*angle
380 FOR I=1 TO pulses:PROCpulse:NEXT I
390 ENDPROC
400 :
410 DEF PROCpulse
420 ?DATREG=(?DATREG OR 8)
430 ?DATREG=(?DATREG AND 247)
440 ENDPROC
```

Commodore 64

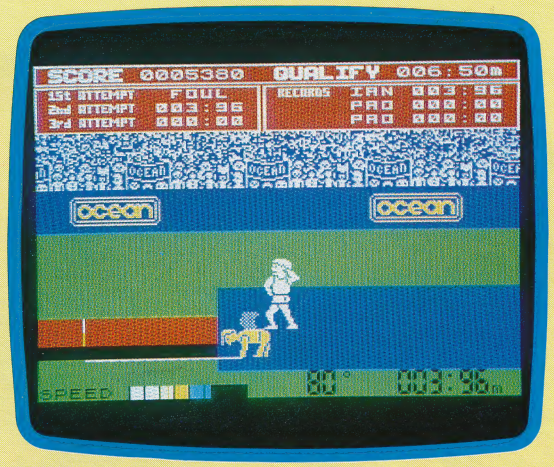
```
10 REM **** CBM LINE FOLLOWER ****
20 :
30 GOSUB2000:REM INIT
40 GOSUB1000:REM FOLLOW LINE
50 END
60 :
1000 REM **** FOLLOW LINE ****
1010 DR=FW:DS=1:GOSUB2500:REM MOVE
1020 GOSUB1500:REM TEST LDRS
1030 IF EF<>1 THEN 1000
1040 RETURN
1050 :
1500 REM **** TEST LDRS ****
1510 LV=PEEK(DATREG)AND 48
1520 IF LV=32 THEN DR=RT:DS=5:GOSUB3000:REM TURN
1530 IF LV=16 THEN DR=LF:DS=5:GOSUB3000:REM TURN
1540 IF LV=0 THEN EF=1
1550 RETURN
1560 :
2000 REM **** INIT ****
2010 DDR=56579:DATREG=56577:POKEDDR,15
2020 POKEDATREG,1
2030 FW=4:BW=2:LF=6:RT=0
2040 PD=3.34446:PA=379/90
2050 EF=0:REM ENDFLAG
2060 RETURN
2070 :
2500 REM **** MOVE (DR,DS) ****
2510 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
2520 PL=PD*ABS(DS)
2530 FOR I=1 TO PL:GOSUB 3500:NEXT I
2550 RETURN
2560 :
3000 REM **** TURN (DR,DS) ****
3010 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
3020 PL=PA*ABS(DS)
3030 FOR I=1 TO PL:GOSUB 3500:NEXT I
3040 RETURN
3050 :
3500 REM **** PULSE ****
3510 POKE DATREG,PEEK(DATREG)OR 8
3520 POKE DATREG,PEEK(DATREG)AND 247
3530 RETURN
```

Sinclair Spectrum

```
10 REM **** SPECTRUM LINE FOLLOWER ****
12 REM DELETE LINES 2010,2020,2510,3010 OF CBM
   VERSION
13 REM AND MAKE THESE CHANGES
15 CLEAR 32499:ST=32500:GOSUB 4500
1510 LET NM=48:GO SUB 4000:LET LV=USR ST
3510 OUT 31,DR+9
3520 OUT 31,DR+1
4000 REM **** PERFORM AND ****
4010 POKE ST+1,IN 31
4020 POKE ST+3,NM:RETURN
4500 REM **** MACHINE CODE LOADER ****
4510 FOR I=ST TO ST+8
4520 READ A:POKE I,A
4530 NEXT I
4540 DATA 62,0,14,0,161,6,0,79,201
4550 RETURN
```

JOYSTICK JURY

Virginia Rizla reviews three programs for the Sinclair Spectrum.



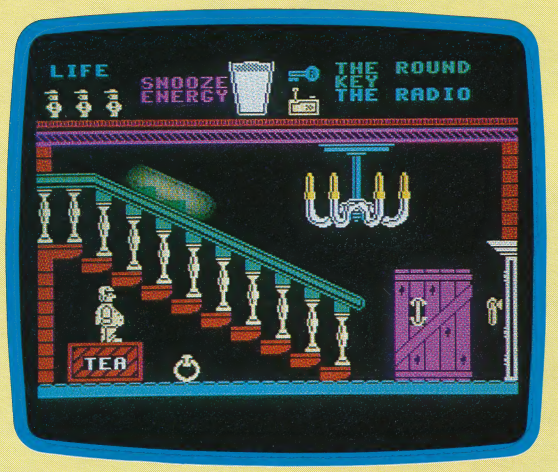
DALEY'S DECATHLON

Ocean, £6.90

Despising anything remotely close to exercise, I was not overcome with joy at the prospect of playing Daley's Decathlon.

The object of the game is, naturally, to compete in the ten events of the decathlon and to score the highest amount of points in each. In the running events, movement is controlled by alternating two keys or else manoeuvering the joystick; after pressing a further key, I found myself leaping great distances and hurling objects with abject skill and grace.

Although tentative for much of the game, I was greatly encouraged by the cheering crowd as I cleared 128 metres in the long jump (beat that, Carl Lewis!). And I didn't feel at all alienated because of the recognisable, and for the most part realistic, graphics and movements. This is one competition I would recommend that you do not boycott.

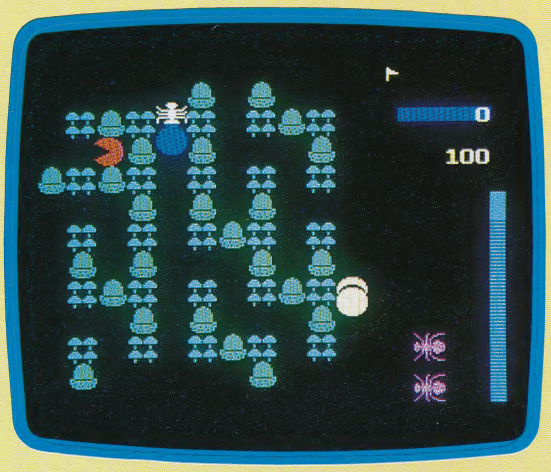


PYJAMARAMA

Mikro-Gen, £6.95

Wally is having a nightmare and proceeds to disrupt several of Jung's theses on somnambulism. (If he is in fact unconscious, then it seems a trifle odd that the goal of the game is to find the key to his alarm clock.) But it is very enjoyable and is in no way offensive to psychoanalysis or dream symbolism.

Several screens represent rooms in Wally's wonderfully drawn and brightly coloured abode. By moving left and right, and jumping on the dubiously tasteful furniture, Wally tries getting through the various doors — by either leaping for the handle or carrying a certain 'object'. Since his life energy is limited, it is preferable to avoid the floating graphics, and the hands that tend to jump out of the floor. Much of the fun was getting the hang of the game, but it continued to provide visual and — more importantly — sporting excitement.



HYPERACTION

Silversoft, £5.95

Silversoft's offering is a well written and colourful dose of hyperactivity. It took me a while to clear the first screen, and upon doing so, found one of the beauties of this game — namely, a new screen means having to devise a new strategy. The first game has you looking for and collecting four 'ZX' symbols in a maze of green shrubbery and mushrooms. By moving about the blocks of foliage, you can effectively block the Pacmen who, unfortunately, cohabitate the maze, and intend to touch you and deplete your life's essence.

The succeeding games are all variations on the maze and get progressively more difficult. It requires much thought and skill, which rules me out as a potential champion, but for those of you wishing to exercise your cerebral bits, this is definitely worth a go.